IBM® DB2 Universal Database™

Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente

Versión 8.2

SC10-3723-01

IBM® DB2 Universal Database™

Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente

Versión 8.2

SC10-3723-01

Antes de utilizar esta información y el producto al que da soporte, asegúrese de leer la información general incluida en el apartado *Avisos*.

Este manual es la traducción del original inglés *IBM DB2 Universal Database Application Development Guide: Programming Client Applications, Version 8.2*, (SC09-4826-01).

Puede realizar pedidos de publicaciones en línea o a través del representante de IBM de su localidad.

- Para realizar pedidos de publicaciones en línea, vaya a IBM Publications Center en www.ibm.com/shop/publications/order
- Para encontrar el representante de IBM correspondiente a su localidad, vaya a IBM Directory of Worldwide Contacts en www.ibm.com/planetwide

Para realizar pedidos de publicaciones en marketing y ventas de DB2 de los EE.UU. o de Canadá, llame al número 1-800-IBM-4YOU (426-4968).

Cuando envía información a IBM, otorga a IBM un derecho no exclusivo para utilizar o distribuir dicha información en la forma en que IBM considere adecuada, sin contraer por ello ninguna obligación con el remitente.

Contenido

Acerca de este manual xv

Parte 1. Introducción 1

Capítulo 1. Visión general de las interfaces de programación soportadas . 3

Herramientas de DB2 Universal Database para desarrollar aplicaciones 3
| Visión general de IBM DB2 Development Add-In . . 4
Interfaces de programación soportadas 5
Interfaces de programación que reciben soporte de DB2 5
Interfaces de programación de aplicaciones de DB2 7
SQL incorporado 8
Interfaz de nivel de llamada de DB2 9
CLI de DB2 frente a SQL dinámico incorporado 10
Java Database Connectivity (JDBC) 12
SQL incorporado para Java (SQLJ). 13
Objetos de datos ActiveX y Objetos de datos remotos. 13
DBI Perl 14
Herramientas de usuario final de ODBC. . . . 15
| DB2 .NET Data Provider 15
Aplicaciones Web 16
Herramientas para crear aplicaciones Web . . . 16
WebSphere Studio 16
XML Extender 17
Habilitación de MQSeries. 17
Net.Data 18
Funciones de programación 18
Funciones de programación de DB2 18
Procedimientos almacenados de DB2 19
Métodos y funciones definidas por el usuario de DB2 20
Centro de desarrollo 20
Tipos definidos por el usuario (UDT) y objetos grandes (LOB) 22
Rutinas de automatización de OLE 23
Funciones de tabla de OLE DB 23
Activadores de DB2 24

Capítulo 2. Codificación de una aplicación DB2 27

Requisitos previos para programación 27
Visión general de la codificación de aplicaciones DB2 28
Programación de una aplicación autónoma . . . 28
Creación de una sección de declaración de una aplicación autónoma 29
Declaración de variables que interactúan con el gestor de bases de datos 29
Declaración de variables que representan objetos de SQL 30
Declaración de variables del lenguaje principal con el Generador de declaraciones db2dclgn . . 32
Relación de variables del lenguaje principal con una sentencia de SQL 33
Declaración de la SQLCA para el manejo de errores 34
Manejo de errores utilizando la sentencia WHENEVER 35
Adición de sentencias no ejecutables a una aplicación 36
Conexión de una aplicación con una base de datos 36
Codificación de transacciones 37
Finalización de una transacción con la sentencia COMMIT 38
Finalización de una transacción con la sentencia ROLLBACK 39
Finalización de un programa de aplicación . . . 40
Finalización implícita de una transacción en una aplicación autónoma 40
Infraestructura de seudocódigo de aplicación . . 41
Recursos para crear prototipos de sentencias de SQL 42
API administrativas en SQL incorporado o en programas de CLI de DB2 43
Control de valores de datos y relaciones. 44
Control de valores de datos 44
Control de valores de datos mediante tipos de datos 44
Control de valores de datos mediante restricciones exclusivas 45
Control de valores de datos mediante restricciones de comprobación de tabla 45
Control de valores de datos mediante restricciones de integridad referencial. 45
Control de valores de datos mediante vistas con la opción Check 46
Control de valores de datos mediante lógica de aplicación y tipos de variables de programa . . 46
Control de relaciones de datos 46
Control de relaciones de datos mediante restricciones de integridad referencial. 47
Control de relaciones de datos mediante activadores 47
Control de relaciones de datos mediante activadores anteriores 48
Control de relaciones de datos mediante activadores posteriores 48
Control de relaciones de datos mediante lógica de aplicación 49
Lógica de aplicación en el servidor 49
Consideraciones sobre autorización para SQL y API 50
Consideraciones sobre autorización para SQL incorporado 50
Consideraciones sobre autorización para SQL dinámico 51

Consideraciones sobre autorización para SQL estático 52
Consideraciones sobre autorización para API . . 53
Prueba de la aplicación 53
Configuración del entorno de prueba para una aplicación 53
Depuración y optimización de una aplicación . . 57

Parte 2. SQL incorporado 59

Capítulo 3. Visión general sobre SQL incorporado 61
Incorporación de sentencias de SQL en un lenguaje principal 61
Creación y preparación del archivo fuente 63
Paquetes, vinculación y SQL incorporado 65
Creación de paquetes para SQL incorporado . . 65
Precompilación de archivos fuente que contienen SQL incorporado 67
Requisitos del archivo fuente para aplicaciones de SQL incorporado 69
Compilación y enlace de archivos fuente que contienen SQL incorporado 70
Creación de paquetes mediante el mandato BIND 71
Creación de versiones de paquetes 72
Efecto de registros especiales en SQL dinámico vinculado 73
El registro especial CURRENT PACKAGE PATH para esquemas de paquete 73
Resolución de nombres de tabla no calificados. . 76
Consideraciones adicionales cuando se vincula 77
Ventajas de la vinculación diferida. 78
Contenido del archivo de vinculación. 78
Relaciones entre aplicación, archivo de vinculación y paquete 79
Indicaciones horarias generadas por el precompilador 79
Revinculación de paquetes 81

Capítulo 4. Cómo escribir programas de SQL estático 83
Características de SQL estático y razones para utilizarlo 83
Ventajas del SQL estático 84
Programa de SQL estático de ejemplo 85
Recuperación de datos en programas de SQL estático 86
Efectos de REOPT en el SQL estático 87
Variables del lenguaje principal en programas de SQL estático 87
Variables del lenguaje principal en SQL estático 87
Declaración de variables del lenguaje principal en programas de SQL estático 89
Cómo hacer referencia a variables del lenguaje principal en programas de SQL estático 90
Variables de indicador en programas de SQL estático 91
Inclusión de variables de indicador en programas de SQL estático 91
Tipos de datos para variables de indicador en programas de SQL estático 93
Ejemplo de variable de indicador en un programa de SQL estático 95
Selección de varias filas mediante un cursor . . . 97
Selección de varias filas utilizando un cursor . . 97
Declaración y utilización de cursores en programas de SQL estático 97
Consideraciones sobre tipos de cursor y unidad de trabajo 98
Ejemplo de un cursor en un programa de SQL estático 100
Manipulación de datos recuperados 102
Actualización y supresión de datos recuperados en programas de SQL estático 102
Tipos de cursor. 102
Ejemplo de captación en un programa de SQL estático 103
Desplazamiento por datos recuperados y manipulación de los mismos 104
Desplazamiento por datos recuperados previamente. 104
Cómo conservar una copia de los datos . . . 105
Recuperación de datos por segunda vez . . . 105
Diferencias en el orden de filas entre la primera y la segunda tabla de resultados 106
Colocación de un cursor al final de una tabla 107
Actualización de datos recuperados previamente 108
Ejemplo de inserción, actualización y supresión en un programa de SQL estático 108
Información de diagnóstico. 110
Códigos de retorno 110
Información de error en los campos SQLCODE, SQLSTATE y SQLWARN 110
Truncamiento de símbolos en la estructura SQLCA 111
Consideraciones sobre el manejador de excepciones, señales e interrupciones 111
Consideraciones sobre las rutinas de lista de salida 112
Recuperación de mensajes de error en una aplicación 112

Capítulo 5. Cómo escribir programas de SQL dinámico 115
Características y razones para utilizar SQL dinámico 115
Razones para utilizar SQL dinámico 115
Sentencias de soporte de SQL dinámico . . . 116
SQL dinámico frente a SQL estático 116
Cursores en programas de SQL dinámico 119
Declaración y utilización de cursores en programas de SQL dinámico 119
Ejemplo de un cursor en un programa de SQL dinámico 120
Efectos de REOPT en el SQL dinámico 121
Efecto de la opción de vinculación DYNAMICRULES en SQL dinámico. 122
La SQLDA en programas de SQL dinámico . . . 124
Variables del lenguaje principal en la SQLDA en programas de SQL dinámico 124

Declaración de la estructura SQLDA en un programa de SQL dinámico 125
Preparación de una sentencia de SQL dinámico utilizando la estructura SQLDA mínima . . . 126
Asignación de una SQLDA con suficientes entradas SQLVAR para un programa de SQL dinámico 128
Descripción de una sentencia SELECT en un programa de SQL dinámico 129
Adquisición de almacenamiento para albergar una fila 129
Proceso del cursor en un programa de SQL dinámico 130
Asignación de una estructura SQLDA para un programa de SQL dinámico 131
Transferencia de datos en un programa de SQL dinámico mediante una estructura SQLDA . . 134
Proceso de sentencias interactivas de SQL en programas de SQL dinámico 135
Determinación del tipo se sentencia en programas de SQL dinámico 135
Proceso de sentencias SELECT de lista de variables en programas de SQL dinámico . . . 136
Cómo guardar peticiones de SQL procedentes de usuarios finales. 137
Marcadores de parámetros en programas de SQL dinámico 137
Cómo proporcionar entrada de variables a SQL dinámico mediante marcadores de parámetros . 137
Ejemplo de marcadores de parámetros en un programa de SQL dinámico 138
Comparación entre Interfaz de nivel de llamada (CLI) de DB2 y SQL dinámico 140
La Interfaz de nivel de llamada (CLI) de DB2 y el SQL dinámico incorporado 140
Ventajas de CLI de DB2 sobre el SQL incorporado 141
Cuándo utilizar CLI de DB2 o SQL incorporado 143

Capítulo 6. Programación en C y C++ 145

Consideraciones sobre programación en C/C++ 145
Secuencias tri-grafo para C y C++ 146
Archivos de entrada y de salida para C y C++ . . 146
Archivos include 147
Archivos include para C y C++ 147
Archivos include en C y C++ 149
Sentencias de SQL incorporado en C y C++ . . . 150
Variables del lenguaje principal en C y C++ . . . 152
Variables del lenguaje principal en C y C++ . . 152
Nombres de variables del lenguaje principal en C y C++ 153
Declaraciones de variables del lenguaje principal en C y C++ 154
Sintaxis de las variables numéricas del lenguaje principal en C y C++. 154
Sintaxis de las variables del lenguaje principal de tipo carácter fijas y terminadas en nulo en C y C++ 156
Sintaxis de las variables del lenguaje principal de tipo carácter de longitud variable en C o C++ 157
Variables de indicador en C y C++ 158
Variables gráficas del lenguaje principal en C y C++ 158
Sintaxis de la declaración gráfica de formatos gráficos de un solo gráfico y terminados en nulo en C y C++ 159
Sintaxis para la declaración gráfica del formato estructurado VARGRAPHIC en C o C++ . . . 160
Sintaxis de las variables del lenguaje principal de objeto grande (LOB) en C o C++ 161
Sintaxis de las variables del lenguaje principal de localizador de objeto grande (LOB) en C o C++ 163
Sintaxis de declaraciones de variables del lenguaje principal de referencia de archivos en C o C++ 164
Inicialización de variables del lenguaje principal en C y C++ 165
Expansión de macros en C 165
Soporte de estructuras del lenguaje principal en C y C++ 166
Tablas de indicadores en C y C++ 168
Series terminadas en nulo en C y C++ 169
Variables del lenguaje principal utilizadas como tipos de datos de puntero en C y C++ 171
Miembros de datos de clase utilizados como variables del lenguaje principal en C y C++ . . 171
Operadores de calificación y de miembro en C y C++ 173
Codificación de caracteres de varios bytes en C y C++ 173
Tipos de datos wchar_t y sqldbchar en C y C++ 174
Opción del precompilador WCHARTYPE en C y C++ 174
Consideraciones sobre EUC en japonés o chino tradicional y UCS-2 en C y C++ 177
Sección declare de SQL con variables del lenguaje principal para C y C++ 178
Consideraciones sobre tipos de datos para C y C++ 179
Tipos de datos SQL soportados en C y C++ . . 180
FOR BIT DATA en C y C++ 184
Tipos de datos C y C++ para procedimientos, funciones y métodos 184
Variables SQLSTATE y SQLCODE en C y C++ . . 186

Capítulo 7. Acceso a bases de datos de varias hebras en aplicaciones C y C++ 187

Objetivo del acceso a base de datos de varias hebras. 187
Recomendaciones para utilizar varias hebras . . . 188
Consideraciones sobre página de códigos y código de país/región para aplicaciones UNIX de varias hebras. 189
Resolución de problemas de aplicaciones de varias hebras. 190
Problemas potenciales con varias hebras . . . 190
Cómo evitar puntos muertos para varios contextos 190

Capítulo 8. Programación en COBOL 193

Consideraciones sobre la programación en COBOL 193
Restricciones de lenguaje en COBOL 193
Acceso a bases de datos de varias hebras en COBOL 193
Archivos de entrada y salida para COBOL . . . 194
Archivos include para COBOL 194
Sentencias de SQL incorporado en COBOL . . . 196
Variables del lenguaje principal en COBOL . . . 198
- Variables del lenguaje principal en COBOL . . 198
- Nombres de variables del lenguaje principal en COBOL 199
- Declaraciones de variables del lenguaje principal en COBOL 200
- Sintaxis de las variables numéricas del lenguaje principal en COBOL 200
- Sintaxis de las variables del lenguaje principal de tipo carácter de longitud fija en COBOL . . 201
- Sintaxis de las variables gráficas del lenguaje principal de longitud fija en COBOL 203
- Variables de indicador en COBOL 204
- Sintaxis de las variables del lenguaje principal LOB en COBOL 204
- Sintaxis de las variables del lenguaje principal de localizador de LOB en COBOL 205
- Sintaxis de las variables del lenguaje principal de referencia de archivos en COBOL 206
- Soporte de estructura del lenguaje principal en COBOL 206
- Tablas de indicadores en COBOL. 208
- REDEFINES en elementos de datos de grupos COBOL 209
- Sección declare de SQL con variables del lenguaje principal para COBOL 209

Consideraciones sobre tipos de datos para COBOL 210
- Tipos de datos de SQL soportados en COBOL 210
- Tipos de datos BINARY/COMP-4 de COBOL 213
- FOR BIT DATA en COBOL. 213

Variables SQLSTATE y SQLCODE en COBOL . . 213
Consideraciones sobre EUC en japonés o chino tradicional y UCS-2 para COBOL. 213
COBOL orientado a objetos. 214

Capítulo 9. Programación en FORTRAN 215

Consideraciones sobre la programación en FORTRAN 215
Restricciones del lenguaje en FORTRAN 215
- Llamada por referencia en FORTRAN 215
- Líneas de depuración y de comentario en FORTRAN 216
- Consideraciones sobre la precompilación en FORTRAN 216
- Acceso a bases de datos de varias hebras en FORTRAN 216

Archivos de entrada y salida para FORTRAN . . 216
Archivos include 216
- Archivos include para FORTRAN 216
- Archivos include en aplicaciones FORTRAN . . 219

Sentencias de SQL incorporado en FORTRAN . . 219
Variables del lenguaje principal en FORTRAN . . 221
- Variables del lenguaje principal en FORTRAN 221
- Nombres de variables del lenguaje principal en FORTRAN 221
- Declaraciones de variables del lenguaje principal en FORTRAN 222
- Sintaxis de las variables numéricas del lenguaje principal en FORTRAN 222
- Sintaxis de las variables de tipo carácter del lenguaje principal en FORTRAN 223
- Variables de indicador en FORTRAN 224
- Sintaxis de las variables del lenguaje principal de objeto grande (LOB) en FORTRAN 225
- Sintaxis de las variables del lenguaje principal de localizador de objeto grande (LOB) en FORTRAN 226
- Sintaxis de las variables del lenguaje principal de referencia de archivos en FORTRAN . . . 226
- Sección declare de SQL con variables del lenguaje principal para FORTRAN 227

Tipos de datos SQL soportados en FORTRAN . . 227
Consideraciones sobre juegos de caracteres de varios bytes en FORTRAN 229
Consideraciones sobre EUC en japonés o chino tradicional y UCS-2 para FORTRAN. 229
Variables SQLSTATE y SQLCODE en FORTRAN 229

Parte 3. ADO.NET, OLE DB y ODBC 231

Capítulo 10. DB2 .NET Data Provider 233

Visión general de DB2 .NET Data Provider . . . 233
Requisitos del sistema DB2 .NET Data Provider 233
Programación de aplicaciones para utilizar DB2 .NET Data Provider 234
- Conexión a una base de datos desde una aplicación utilizando DB2 .NET Data Provider . 234
- Ejecución de sentencias de SQL desde una aplicación utilizando DB2 .NET Data Provider . 234
- Lectura de conjuntos de resultados de una aplicación utilizando DB2 .NET Data Provider . 235
- Invocación de procedimientos almacenados desde una aplicación utilizando DB2 .NET Data Provider 236
- Tipos de datos SQL soportados para DB2 .NET Data Provider 238

Capítulo 11. IBM OLE DB Provider para DB2 241

Objetivo de IBM OLE DB Provider para DB2. . . 241
Tipos de aplicaciones soportados por IBM OLE DB Provider para DB2. 242
Servicios OLE DB 242
- Modelo de hebras soportado por IBM OLE DB Provider 242
- Manipulación de objetos grandes con IBM OLE DB Provider. 243
- Conjuntos de filas de esquema soportados por IBM OLE DB Provider 243

Habilitación automática de servicios OLE DB por parte de IBM OLE DB Provider 245
Servicios de datos 245
Modalidades de cursor soportadas en IBM OLE DB Provider 245
Correlaciones de tipos de datos entre DB2 y OLE DB 245
Conversión de datos para establecer datos de tipos OLE DB en tipos DB2. 246
Conversión de datos para establecer datos de tipos DB2 en tipos OLE DB. 248
Restricciones de IBM OLE DB Provider. 250
Soporte de IBM OLE DB Provider de interfaces y componentes de OLE DB 250
Soporte de IBM OLE DB Provider para propiedades de OLE DB. 253
Conexiones a fuentes de datos mediante IBM OLE DB Provider 255
Aplicaciones ADO. 256
Palabras clave de series de conexión de ADO 256
Conexiones con fuentes de datos con aplicaciones ADO Visual Basic. 257
Cursores desplazables actualizables en aplicaciones ADO 257
Limitaciones de las aplicaciones ADO 257
Soporte de IBM OLE DB Provider de propiedades y métodos ADO 258
Aplicaciones C y C++ 261
Compilación y enlace de aplicaciones C/C++ e IBM OLE DB Provider 261
Conexiones con fuentes de datos en aplicaciones C/C++ mediante IBM OLE DB Provider . . . 262
Transacciones distribuidas MTS y COM+ 262
Soporte de transacciones distribuidas MTS y COM+ e IBM OLE DB Provider 262
Habilitación del soporte de MTS en DB2 Universal Database para aplicaciones C/C++. . 263

Capítulo 12. OLE DB .NET Data Provider 265
OLE DB .NET Data Provider 265
Restricciones de OLE DB .NET Data Provider . . 266
Agrupación de conexiones en aplicaciones de OLE DB .NET Data Provider 270
Columnas de tiempo en aplicaciones de OLE DB .NET Data Provider 271
Objetos ADORecordset en aplicaciones de OLE DB .NET Data Provider 272

Capítulo 13. ODBC .NET Data Provider 273
ODBC .NET Data Provider 273
Restricciones de ODBC .NET Data Provider . . . 273

Parte 4. Java 281

Capítulo 14. Introducción al soporte de aplicaciones Java 283

Capítulo 15. Programación de aplicaciones JDBC 287
Conceptos básicos de la programación de aplicaciones JDBC 287
Pasos básicos para escribir una aplicación JDBC 287
Paquetes Java para el soporte JDBC 290
Variables en aplicaciones JDBC 290
Conexión de las aplicaciones JDBC a una fuente de datos 291
Conexión de aplicaciones DB2 a una fuente de datos utilizando la interfaz DriverManager con el Controlador JDBC de DB2 de tipo 2 292
Conexión a una fuente de datos utilizando la interfaz DriverManager con el Controlador JDBC de DB2 Universal 294
Conexión a una fuente de datos utilizando la interfaz DataSource 297
Establecimiento del nivel de aislamiento para una transacción de JDBC 299
Objetos de conexión JDBC 299
Confirmación o retrotracción de transacciones JDBC 300
Cierre de una conexión con una fuente de datos JDBC 300
Interfaces JDBC para ejecutar SQL 300
Utilización del método Statement.executeUpdate para crear y modificar objetos DB2 301
Utilización del método Statement.executeQuery para recuperar datos de tablas DB2 302
Uso del método PreparedStatement.executeUpdate para actualizar datos de tablas DB2. 303
Uso del método PreparedStatement.executeQuery para obtener datos de DB2 305
Utilización de métodos CallableStatement para llamar a procedimientos almacenados 306
Manejo de una excepción de SQL cuando se utiliza el Controlador JDBC de DB2 Universal . 308
Manejo de una excepción de SQL cuando se utiliza el Controlador JDBC de DB2 de tipo 2. . 311
Manejo de un aviso de SQL cuando se utiliza el Controlador JDBC de DB2 Universal 312
Manejo de un aviso de SQL cuando se utiliza el Controlador JDBC de DB2 de tipo 2 313
Conceptos avanzados de la programación de aplicaciones JDBC 315
Datos LOB en aplicaciones JDBC con el Controlador JDBC de DB2 Universal 315
Tipos de datos Java para recuperar o actualizar datos de columnas LOB en aplicaciones JDBC . 316
Uso de valores ROWID en JDBC con el Controlador JDBC universal de DB2. 318
Tipos diferenciados en aplicaciones JDBC . . . 319
Puntos de rescate en aplicaciones JDBC. . . . 320
Recuperación de valores de columnas de identidad en aplicaciones JDBC 321
Recuperación de varios conjuntos de resultados de un procedimiento almacenado en una aplicación JDBC 323

Uso de ResultSetMetaData para obtener información sobre un conjunto de resultados . . 326
Uso de DatabaseMetaData para obtener información sobre una fuente de datos 327
Uso de ParameterMetaData para obtener información sobre los parámetros de un objeto PreparedStatement 329
Realización de actualizaciones por lotes en aplicaciones JDBC 330
Recuperación de información de una excepción BatchUpdateException 332
Características de un conjunto de resultados de JDBC cuando se utiliza el Controlador JDBC de DB2 Universal 334
Especificación de las capacidades de actualización, desplazamiento y retención para conjuntos de resultados en aplicaciones de JDBC 335
Creación y despliegue de objetos DataSource 338
Soporte de redireccionamiento del cliente del Controlador JDBC universal de DB2. 339
Suministro de información ampliada sobre el cliente al servidor DB2 con el Controlador JDBC universal de DB2 341

Capítulo 16. Programación de aplicaciones SQLJ 343

Conceptos básicos de la programación de aplicaciones SQLJ 343
Pasos básicos para escribir una aplicación SQLJ 343
Paquetes Java para el soporte SQLJ 346
Variables en aplicaciones SQLJ 347
Comentarios en una aplicación SQLJ 348
Conexión a una fuente de datos utilizando SQLJ 349
Establecimiento del nivel de aislamiento para una transacción de SQLJ. 354
Confirmación o retrotracción de transacciones SQLJ 354
Puntos de rescate en aplicaciones SQLJ. 354
Cierre de la conexión con una fuente de datos en una aplicación SQLJ 356
Sentencias de SQL en una aplicación SQLJ . . 356
Creación y modificación de objetos DB2 en una aplicación SQLJ. 357
Recuperación de datos de tablas DB2 por una aplicación SQLJ. 357
Utilización de un iterador de nombre en una aplicación SQLJ. 358
Utilización de un iterador de posición en una aplicación SQLJ. 361
Ejecución de operaciones UPDATE y DELETE de posición en una aplicación SQLJ 363
Varios iteradores abiertos para una misma sentencia de SQL en una aplicación SQLJ . . . 368
Uso de varias instancias abiertas de un iterador en una aplicación SQLJ 369
Invocación de procedimientos almacenados en una aplicación SQLJ 370
Manejo de errores de SQL en una aplicación SQLJ 371
Manejo de avisos de SQL en una aplicación SQLJ 372
Conceptos avanzados de la programación de aplicaciones SQLJ 373
Uso de SQLJ y JDBC en la misma aplicación 373
Datos LOB en aplicaciones JDBC con el Controlador JDBC de DB2 Universal 376
Tipos de datos Java para recuperar o actualizar datos de columnas LOB en aplicaciones SQLJ . 376
Uso de valores ROWID en JDBC con el Controlador JDBC universal de DB2. 379
Tipos diferenciados en aplicaciones SQLJ . . . 381
Control de la ejecución de sentencias de SQL en SQLJ 381
Recuperación de varios conjuntos de resultados de un procedimiento almacenado en una aplicación SQLJ. 382
Realización de actualizaciones por lotes en aplicaciones SQLJ 383
Uso de iteradores pasados como variables en operaciones UPDATE o DELETE de posición de una aplicación SQLJ 387
Utilización de iteradores desplazables en una aplicación SQLJ. 389

Capítulo 17. Información de consulta sobre JDBC y SQLJ 395

Tipos de datos de Java, JDBC y SQL. 395
Propiedades del Controlador JDBC de DB2 Universal. 400
Comparación del soporte de controlador para las API de JDBC 407
Información de consulta sobre sentencias de SQLJ 427
Cláusula SQLJ 427
Expresión de lenguaje principal de SQLJ . . . 427
Cláusula implements de SQLJ 428
Cláusula with de SQLJ 429
Cláusula connection-declaration de SQLJ . . . 430
Cláusula de declaración de iterador de SQLJ 431
Cláusula ejecutable de SQLJ 433
Cláusula context de SQLJ 434
Cláusula de sentencia de SQLJ 434
Cláusula SET TRANSACTION de SQLJ . . . 436
Cláusula de asignación de SQLJ 437
Cláusula de conversión a iterador de SQLJ . . 438
Clases e interfaces selectas de sqlj.runtime. . . . 438
Información de consulta sobre el Controlador JDBC universal de DB2 446
Resumen de las extensiones del Controlador JDBC de DB2 Universal para JDBC 446
Diferencias de JDBC entre el Controlador JDBC de DB2 Universal y otros controladores JDBC de DB2 459
Diferencias respecto a SQLJ entre el Controlador JDBC universal de DB2 y otros controladores JDBC de DB2 466
Códigos de error emitidos por el Controlador JDBC de DB2 Universal 468
Estados de SQL emitidos por el Controlador JDBC de DB2 Universal 469

Capítulo 18. Instalación de los controladores JDBC 471
Instalación del Controlador JDBC de DB2 Universal 471

Capítulo 19. Seguridad en JDBC y SQLJ 477
Seguridad cuando se utiliza el Controlador JDBC de DB2 de tipo 2 477
Seguridad cuando se utiliza el Controlador JDBC de DB2 Universal 478
Seguridad basada en ID de usuario y contraseña cuando se utiliza el Controlador JDBC de DB2 Universal. 479
Seguridad basada solo en el ID de usuario cuando se utiliza el Controlador JDBC de DB2 Universal . 480
Seguridad mediante ID de usuario cifrado o contraseña cifrada con el Controlador JDBC de DB2 Universal 481
Seguridad Kerberos con el Controlador JDBC de DB2 Universal 482

Capítulo 20. Diagnóstico de problemas de JDBC y SQLJ 487
Diagnóstico de problemas de JDBC y SQLJ cuando se utiliza el Controlador JDBC universal de DB2 . 487
Diagnóstico de problemas de JDBC y SQLJ con el Controlador JDBC de DB2 Universal. 487
Ejemplo de rastreo para el Controlador JDBC de DB2 Universal 489
Diagnóstico de problemas de JDBC y SQLJ cuando se utiliza el Controlador JDBC de DB2 de tipo 2. . 494
Recurso de rastreo de CLI/ODBC/JDBC . . . 494
Archivos de rastreo de CLI y JDBC 500

Capítulo 21. Java 2 Platform Enterprise Edition 511
Visión general de Java 2 Platform Enterprise Edition (J2EE) 511
Java 2 Platform Enterprise Edition 511
Contenedores de Java 2 Platform Enterprise Edition 512
Servidor Java 2 Platform Enterprise Edition . . . 513
Requisitos de bases de datos de Java 2 Enterprise Edition 513
Java Naming and Directory Interface (JNDI) . . . 513
Gestión de transacciones Java 514
Ejemplo de una transacción distribuida que utiliza métodos de JTA 515
Enterprise Java Beans. 520

Parte 5. Otras interfaces de programación 523

Capítulo 22. Programación en Perl 525
Consideraciones sobre la programación en Perl . . 525
Restricciones de Perl 525
Acceso a bases de datos de varias hebras en Perl 525
Conexiones de bases de datos en Perl 525
Captación de resultados en Perl 526
Marcadores de parámetros en Perl 527
Variables SQLSTATE y SQLCODE en Perl 527
Ejemplo de programa Perl 527

Capítulo 23. Programación en REXX 529
Consideraciones sobre la programación en REXX 529
Restricciones del lenguaje en REXX 530
Restricciones del lenguaje en REXX 530
Registro de SQLEXEC, SQLDBS y SQLDB2 en REXX 530
Acceso a bases de datos de varias hebras en REXX 531
Consideraciones sobre EUC en japonés o chino tradicional para REXX 531
SQL incorporado en aplicaciones REXX. 531
Variables del lenguaje principal en REXX 533
Variables del lenguaje principal en REXX . . . 533
Nombres de variables del lenguaje principal en REXX 534
Referencias a variables del lenguaje principal en REXX 534
Variables de indicador en REXX 534
Variables de REXX predefinidas 534
Variables del lenguaje principal de LOB en REXX 536
Sintaxis de las declaraciones de localizador de LOB en REXX 537
Sintaxis de las declaraciones de referencia de archivos LOB en REXX 537
Borrado de variables del lenguaje principal de LOB en REXX 538
Cursores en REXX. 539
Tipos de datos SQL soportados en REXX 539
Requisitos de ejecución para REXX 541
Creación y ejecución de aplicaciones REXX . . 541
Archivos de vinculación de REXX 542
Sintaxis de las API para REXX. 542
Llamada a procedimientos almacenados desde REXX 544
Procedimientos almacenados en REXX 544
Llamadas a procedimientos almacenados en REXX 544
Consideraciones del cliente para llamar a procedimientos almacenados en REXX 545
Consideraciones del servidor para llamar a procedimientos almacenados en REXX 546
Recuperación de valores de precisión y escala (SCALE) de campos decimales de la SQLDA . . 546

Capítulo 24. Escritura de aplicaciones utilizando funciones de DB2 WebSphere MQ 547
Visión general funcional de WebSphere MQ . . . 547
Mensajería de WebSphere MQ. 549
Envío de mensajes con funciones de WebSphere MQ. 552
Recuperación de mensajes con funciones de WebSphere MQ. 553

Conectividad de aplicación a aplicación de WebSphere MQ. 555
Comunicaciones de solicitud/respuesta con funciones de WebSphere MQ 557
Publicación/suscripción con funciones de WebSphere MQ. 558

Capítulo 25. WebSphere 563
Conexión con datos de la empresa 563
Fuentes de datos y agrupación de conexiones WebSphere 563
Ventajas de la agrupación de conexiones de WebSphere 564
Colocación de sentencias en antememoria en WebSphere 565

Parte 6. Plug-ins de seguridad . . 567

Capítulo 26. Plug-ins de seguridad 569
Plug-ins de seguridad 569
Ubicaciones de las bibliotecas de plug-in de seguridad 573
Convenios para nombrar plug-ins de seguridad 574
Soporte de plug-in de seguridad para los ID de usuario de dos componentes 575
Consideraciones sobre los sistemas de 32 y 64 bits para los plug-ins de seguridad 578
Determinación de problemas para plug-ins de seguridad 578
Despliegue de un plug-in de recuperación de grupos 580
Despliegue de un plug-in de ID de usuario/contraseña 581
Despliegue de un plug-in de GSS-API 583
Despliegue de un plug-in Kerberos 584

Capítulo 27. Desarrollo de plug-ins de seguridad 587
Carga de plug-ins de seguridad por DB2 587
Restricciones para bibliotecas de plug-in de seguridad 588
Códigos de retorno de los plug-ins de seguridad 590
Mensajes de error de los plug-ins de seguridad . . 593
Secuencias de llamada para las API de plug-in de seguridad 594

Capítulo 28. Las API de plug-in de seguridad 597
Las API de plug-in de seguridad 597
Las API de plug-in de grupo 599
Las API de plug-ins de recuperación de grupos 599
db2secGroupPluginInit - Inicializar plug-in de grupo 601
db2secPluginTerm - Depurar recursos de plug-in de grupo 602
db2secGetGroupsForUser - Obtener la lista de grupos del usuario 602
db2secDoesGroupExist - Comprobar si existe el grupo 606
db2secFreeGroupListMemory - Liberar memoria de lista de grupo 607
db2secFreeErrormsg - Liberar la memoria de mensajes de error 608
Las API de plug-in de autenticación de usuario 608
Las API del plug-in de autenticación por ID usuario/contraseña 608
db2secClientAuthPluginInit - Inicializar el plug-in de autenticación del cliente 615
db2secClientAuthPluginTerm - Depurar recursos de plug-in de autenticación de cliente 616
db2secRemapUserid - Reasignar ID de usuario y contraseña 617
db2secGetDefaultLoginContext - Obtener contexto de conexión por omisión 618
db2secGenerateInitialCred - Generar credenciales iniciales 620
db2secValidatePassword - Validar contraseña 622
db2secProcessServerPrincipalName - Procesar nombre de principal de servicio devuelto desde servidor 624
db2secFreeToken - Liberar memoria retenida por símbolo (token). 625
db2secFreeInitInfo - Liberar recursos retenidos por db2secGenerateInitialCred() 626
db2secServerAuthPluginInit - Inicializar el plug-in de autenticación del servidor 627
db2secServerAuthPluginTerm - Depurar recursos de plug-in de autenticación de servidor 629
db2secGetAuthIDs - Obtener los ID de autenticación 629
db2secDoesAuthIDExist - Comprobar si existe el ID de autenticación 631
Las API de plug-in de GSS-API 632
Las API y definiciones necesarias para los plug-ins de autenticación de GSS-API 632
Restricciones para los plug-in de autenticación de GSS-API 633
Creación de versiones de la API de plug-in de seguridad 634

Parte 7. Conceptos generales sobre aplicaciones DB2. 635

Capítulo 29. Soporte de idiomas nacionales 637
Visión general del orden de clasificación 637
Ordenes de clasificación 637
Comparaciones de caracteres basadas en ordenes de clasificación 639
Comparaciones que no dependen de mayúsculas y minúsculas mediante la función TRANSLATE 640
Diferencias entre los órdenes de clasificación de EBCDIC y ASCII 641
Orden de clasificación especificado al crear una base de datos 642
Órdenes de clasificación de ejemplo 644
Páginas de códigos y entornos locales 645
Derivación de valores de página de códigos . . 645

Derivación de entornos locales en programas de aplicación 645
Cómo DB2 deriva entornos locales 646
Consideraciones sobre aplicaciones 646
Consideraciones sobre el soporte de idiomas nacionales y sobre el desarrollo de aplicaciones . 646
Soporte de idiomas nacionales y sentencias de SQL 648
Rutinas remotas 649
Consideraciones sobre nombres de paquetes en entornos de páginas de códigos combinadas . . 649
Página de códigos activa para precompilación y vinculación 650
Página de códigos activa para la ejecución de aplicaciones 650
Conversión de caracteres entre páginas de códigos diferentes 650
Cuándo se produce conversión de la página de códigos 650
Sustitución de caracteres durante conversiones de páginas de códigos 651
Conversiones de páginas de códigos soportadas 652
Factor de expansión de conversión de página de códigos 653
Juegos de caracteres DBCS 654
Juegos de caracteres de Extended UNIX Code (EUC) 654
Programas CLI, ODBC, JDBC y SQLJ en un entorno DBCS 655
Consideraciones sobre los juegos de códigos EUC y UCS-2 de japonés y chino tradicional 656
Consideraciones sobre los juegos de códigos EUC y UCS-2 de japonés y chino tradicional . . 656
Consideraciones sobre las bases de datos y los clientes de doble byte con EUC mixtos 658
Consideraciones sobre la conversión de caracteres para usuarios de chino tradicional . . 658
Datos gráficos en aplicaciones EUC en japonés o chino tradicional 659
Desarrollo de aplicaciones en situaciones de páginas de códigos distintas 660
Validación de parámetros basada en cliente en un entorno de juegos de códigos mixtos . . . 663
Sentencia DESCRIBE en entornos de juegos de códigos mixtos 664
Datos de longitud fija y de longitud variable en entornos de juegos de códigos mixtos 666
Desbordamiento de longitud de serie de conversión de página de códigos en entornos de juegos de códigos mixtos 666
Aplicaciones conectadas a bases de datos Unicode 668

Capítulo 30. Gestión de transacciones 671
Unidad de trabajo remota 671
Consideraciones sobre la actualización múltiple 671
Actualización múltiple 672
Cuándo utilizar la actualización múltiple . . . 672
Sentencias de SQL en aplicaciones de actualización múltiple 673
Precompilación de aplicaciones de actualización múltiple 674
Consideraciones sobre parámetros de configuración correspondientes a aplicaciones de actualización múltiple 676
Acceso a servidores de sistema principal, AS/400 o iSeries 677
Transacciones simultáneas 678
Transacciones simultáneas 678
Problemas potenciales con transacciones simultáneas 679
Cómo evitar puntos muertos para transacciones simultáneas 679
Puntos de rescate y transacciones. 680
Gestión de transacciones con puntos de rescate 680
Comparación entre puntos de rescate de la aplicación y bloques de SQL compuesto . . . 682
Sentencias de SQL para crear y controlar puntos de rescate 684
Restricciones sobre el uso de puntos de rescate 685
Puntos de rescate y Lenguaje de definición de datos (DDL) 685
Anidamiento de puntos de rescate 686
Puntos de rescate e inserciones en almacenamiento intermedio 687
Puntos de rescate y bloqueo de cursor 687
Puntos de rescate y gestores de transacciones que cumplen con XA 688
Consideraciones sobre la programación de interfaces XA de X/Open 688
Enlace de aplicaciones y la interfaz XA de X/Open 691
Gestión de transacciones MTS y COM+ 691
Microsoft Transaction Server (MTS) y Microsoft Component Services (COM+) como gestor de transacciones 691
Soporte ligeramente agrupado con Microsoft Component Services (COM+) 693
Tiempo de espera de transacciones de Microsoft Transaction Server (MTS) y Microsoft Component Services (COM+) 694
Agrupación de conexiones ODBC y ADO con Microsoft Transaction Server (MTS) y Microsoft Component Services (COM+) 695

Capítulo 31. Consideraciones sobre programación para entornos de bases de datos particionadas 697
Cursores FOR READ ONLY en un entorno de bases de datos particionadas 697
DDS dirigida y elusión local 697
DDS dirigida y elusión local en entornos de bases de datos particionadas 697
DDS dirigida en entornos de bases de datos particionadas 698
Elusión local en entornos de bases de datos particionadas 698
Inserciones colocadas en almacenamiento intermedio 699
Inserciones en almacenamiento intermedio en entornos de bases de datos particionadas . . . 699

Consideraciones sobre la utilización de inserciones en almacenamiento intermedio . . 702
Restricciones en la utilización de inserciones en almacenamiento intermedio 704
Ejemplo de extracción un gran volumen de datos en una base de datos particionada 705
Creación de un entorno simulado de bases de datos particionadas 710
Resolución de problemas 710
Consideraciones sobre el manejo de errores en entornos de bases de datos particionadas . . . 710
Errores graves en entornos de bases de datos particionadas 711
Varias estructuras SQLCA fusionadas 711
Partición que devuelve el error 712
Aplicaciones en bucle o suspendidas 712

Capítulo 32. Técnicas comunes de aplicación de DB2 715
Ejecución de aplicaciones desde Local System Account de Windows. 715
Columnas generadas 715
Columnas de identidad 716
Recuperación de conjuntos de resultados a partir de una sentencia de cambio de datos de SQL. . . 717
Tablas de resultados intermedios 718
Vistas y tablas de destino 719
Ordenación del conjunto de resultados basada en INPUT SEQUENCE 719
Recuperación de conjuntos de resultados a partir de sentencias de cambio de datos de SQL utilizando cursores 720
Incluir columnas 721
Incluir columnas en operaciones INSERT . . . 721
Incluir columnas en operaciones UPDATE and DELETE 722
Operaciones UPDATE, INSERT, DELETE y MERGE de búsqueda frente a selecciones completas . . . 722
Valores secuenciales y objetos de secuencia . . . 723
Generación de valores secuenciales 723
Gestión del comportamiento de secuencias . . 725
Rendimiento de la aplicación y objetos de secuencia 726
Comparación entre objetos de secuencia y columnas de identidad 726
Tablas temporales declaradas y rendimiento de la aplicación 726
Transmisión de grandes volúmenes de datos a través de una red 729

Parte 8. Apéndices 731

Apéndice A. Sentencias de SQL soportadas 733

Apéndice B. Limitaciones en el despliegue de los plug-ins de seguridad 737

Apéndice C. Programación en un entorno de sistema principal o iSeries 739
Aplicaciones en entornos de sistema principal o iSeries 739
Lenguaje de definición de datos en entornos de sistema principal e iSeries 740
Lenguaje de manipulación de datos en entornos de sistema principal e iSeries 741
Lenguaje de control de datos en entornos de sistema principal e iSeries 742
Gestión de conexiones de bases de datos con DB2 Connect 742
Proceso de peticiones de interrupción 743
Atributos de paquete, PREP y BIND. 743
Diferencias de atributos de paquete entre sistemas de bases de datos relacionales de IBM . 743
Opción CNULREQD BIND para series C terminadas en nulo 744
Variables SQLCODE y SQLSTATE autónomas 744
Niveles de aislamiento soportados por DB2 Connect 744
Órdenes de clasificación definidos por el usuario 745
Diferencias en la integridad referencial entre sistemas de bases de datos relacionales de IBM . . 745
Bloqueo y portabilidad de aplicaciones 746
Diferencias en SQLCODE y SQLSTATE entre sistemas de bases de datos relacionales de IBM . . 746
Diferencias en el catálogo del sistema entre sistemas de bases de datos relacionales de IBM . . 747
Desbordamientos por conversión numérica en asignaciones de recuperación 747
Procedimientos almacenados en entornos de sistema principal o iSeries 747
Soporte de DB2 Connect de SQL compuesto . . . 748
Actualización múltiple con DB2 Connect 749
Sentencias de SQL de servidor de sistema principal e iSeries soportadas por DB2 Connect 750
Sentencias de SQL de servidor de sistema principal e iSeries rechazadas por DB2 Connect 750

Apéndice D. Simulación de clasificación binaria EBCDIC. 753

Apéndice E. Información técnica sobre DB2 Universal Database 757
Documentación y ayuda de DB2 757
Actualizaciones de la documentación de DB2 757
Centro de información de DB2 758
Escenarios de instalación del Centro de información de DB2 760
Instalación del Centro de información de DB2 utilizando el asistente de instalación de DB2 (UNIX) 762
Instalación del Centro de información de DB2 utilizando el asistente de instalación de DB2 (Windows) 765
Invocación del Centro de información de DB2 . . 767
Actualización del Centro de información de DB2 instalado en el sistema o en un servidor de intranet 768

| Visualización de temas en el idioma preferido en el
| Centro de información de DB2 769
Documentación PDF e impresa de DB2 770
| Información básica de DB2 770
Información de administración 771
Información para el desarrollo de aplicaciones 771
Información de Business Intelligence 772
Información de DB2 Connect 772
Información de iniciación 773
Información de aprendizaje. 773
Información sobre componentes opcionales . . 774
Notas del release 774
Impresión de manuales de DB2 desde archivos PDF 775
Solicitud de manuales de DB2 impresos 776
Invocación de ayuda según contexto desde una herramienta de DB2 776
| Invocación de la ayuda de mensajes desde el
| procesador de línea de mandatos. 778
| Invocación de la ayuda de mandatos desde el
| procesador de línea de mandatos. 778
| Invocación de la ayuda para estados de SQL desde
| el procesador de línea de mandatos 779
Guías de aprendizaje de DB2 779
Información de resolución de problemas de DB2 780
Accesibilidad 781
Entrada de teclado y navegación 781
Pantalla accesible 781
Compatibilidad con tecnologías de asistencia 782
Documentación accesible 782
| Diagramas de sintaxis en formato decimal con
| puntos. 782
| Certificación Common Criteria de productos DB2
| Universal Database 784

Apéndice F. Avisos 785
Marcas registradas. 787

Índice. 789

Cómo ponerse en contacto con IBM 809
Información sobre productos 809

Acerca de este manual

La *Guía de desarrollo de aplicaciones* es una aplicación en tres volúmenes que describe lo que hay que saber acerca de la codificación, depuración, construcción y ejecución de aplicaciones de DB2:

- La publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente* contiene lo que hay que saber para codificar aplicaciones de DB2 autónomas que se ejecuten en clientes DB2. Incluye información sobre los temas siguientes:
  - Interfaces de programación soportadas por DB2. Se proporcionan descripciones de alto nivel para DB2 Developer's Edition, interfaces de programación soportadas, recursos para crear aplicaciones de Web y características de programación suministradas por DB2, como por ejemplo rutinas y activadores.
  - La estructura general que debe tener una aplicación de DB2. Se indican recomendaciones sobre cómo mantener las relaciones y los valores de los datos en la base de datos, se describen consideraciones sobre autorizaciones y se proporciona información sobre cómo probar y depurar la aplicación.
  - SQL incorporado, tanto dinámico como estático. Se describen las consideraciones generales para el SQL incorporado, así como aspectos específicos que se aplican al uso del SQL estático y dinámico en aplicaciones de DB2.
  - Lenguajes de sistema principal e interpretados soportados, como por ejemplo C/C++, COBOL, Perl y REXX, y el modo de utilizar SQL incorporado en las aplicaciones escritas en estos lenguajes.
  - El Proveedor de datos DB2 .NET, así como los proveedores de datos OLE DB .NET y ODBC .NET.
  - Java (tanto JDBC como SQLJ) y consideraciones para crear aplicaciones de Java que se utilicen en los productos WebSphere Application Server.
  - IBM OLE DB Provider para Servidores DB2. Se proporciona información general sobre el soporte de IBM OLE DB para servicios, componentes y propiedades de OLE DB. Asimismo, se brinda información específica sobre aplicaciones de Visual Basic y Visual C++ que utilizan la interfaz OLE DB para ActiveX Data Objects (ADO).
  - Aspectos de soporte de idiomas nacionales. Se describen temas generales, como por ejemplo las secuencias de clasificación, la deducción de páginas de códigos y las conversiones de caracteres. También se describen aspectos más específicos, como por ejemplo las páginas de códigos DBCS, los juegos de caracteres EUC y aspectos que se aplican a los entornos de japonés y chino tradicional EUC y UCS-2.
  - Gestión de transacciones. Se describen aspectos que afectan a las aplicaciones que realizan actualizaciones para varios sitios, así como a las aplicaciones que realizan transacciones simultáneas.
  - Aplicaciones en entornos de base de datos particionada. Se describen DSS dirigidos, elusiones locales, inserciones en almacenamiento intermedio y resolución de problemas en las aplicaciones que se encuentran en entornos de base de datos particionada.
  - Técnicas de aplicación utilizadas normalmente. Se proporciona información sobre cómo utilizar columnas generadas y de identidad, tablas temporales declaradas y cómo usar puntos de grabación para gestionar transacciones.

  - Las sentencias de SQL cuyo uso se soporta en aplicaciones de SQL incorporado.
  - Aplicaciones que acceden a entornos de sistema principal e iSeries. Se describen los aspectos relativos a las aplicaciones de SQL incorporado que acceden a entornos de sistema principal e iSeries.
  - La simulación de la clasificación binaria EBCDIC.
- La publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor* contiene lo que hay que saber sobre la programación en que se utilizan objetos de la parte del servidor, los cuales incluyen rutinas, objetos grandes, tipos definidos por el usuario y activadores. Incluye información sobre los temas siguientes:
  - Rutinas (procedimientos almacenados, funciones definidas por el usuario y métodos), que incluye:
    - Rendimiento de rutinas, seguridad, consideraciones sobre la gestión de bibliotecas y restricciones.
    - Creación de rutinas, incluidas rutinas externas, y la sentencia CREATE.
    - Modalidades de parámetros de procedimientos y manejo de parámetros.
    - Conjuntos de resultados de procedimientos.
    - Procedimientos de SQL, incluidos la depuración y manejo de condiciones.
    - Funciones de tabla y escalares definidas por el usuario.
    - Llamadas de funciones de tabla y escalares definidas por el usuario (FIRST call, FINAL call,...) y áreas reutilizables.
    - Métodos.
    - Autorizaciones y enlace de rutinas externas.
    - Consideraciones específicas del lenguaje para rutinas de C, de Java, de ejecución en el lenguaje común .NET y de automatización de OLE.
    - Invocación de rutinas.
    - Selección de función.
    - Pase de tipos diferenciados y LOB a las funciones.
    - Páginas de códigos y rutinas.
  - Objetos grandes, que incluyen uso de LOB y localizadores, variables de referencia y datos CLOB.
  - Tipos diferenciados definidos por el usuario, que incluyen tipificación estricta, definición y eliminación de UDT, creación de tablas con tipos estructurados, utilización de tipos diferenciados y tablas con tipo para aplicaciones específicas, manipulación de tipos diferenciados y difusión entre los mismos y realización de comparaciones y asignaciones con tipos diferenciados, que incluyen operaciones UNION sobre columnas con tipo de forma diferenciada.
  - Tipos estructurados definidos por el usuario, que incluyen almacenamiento de instancias y creación de instancias, jerarquías de tipos estructurados, definición del comportamiento de los tipos estructurados, distribución dinámica de métodos, funciones de comparación, difusión y constructor y métodos mutador y observador correspondientes a tipos estructurados.
  - Tablas con tipo, que incluyen creación, eliminación, sustitución y almacenamiento de objetos, definición de identificadores de objetos generados por el sistema y restricciones en las columnas de identificador.
  - Tipos de referencia, que incluyen relaciones entre objetos de tablas con tipo, relaciones semánticas con referencias e integridad referencial frente a referencias de ámbito.

  - Tablas y vistas con tipo, que incluyen tipos estructurados como tipos de columnas, funciones de transformación y grupos de transformación, correlaciones de programas de lenguaje principal y variables del lenguaje principal de tipos estructurados.
  - Desencadenantes, que incluyen los desencadenantes INSERT, UPDATE y DELETE, interacciones con restricciones referenciales, líneas generales de creación, granularidad, hora de activación, tablas y variables de transición, acciones desencadenadas, varios desencadenantes y sinergia entre desencadenantes, restricciones y rutinas.
- La publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones* contiene lo que hay que saber para crear y ejecutar aplicaciones de DB2 en los sistemas operativos soportados por DB2:
  - AIX
  - HP-UX
  - Linux
  - Solaris
  - Windows

  Incluye información sobre los temas siguientes:
  - Software y servidores para crear aplicaciones soportados por DB2, que incluye los compiladores e intérpretes soportados.
  - Los archivos de programas de ejemplo de DB2, makefiles, archivos de creación y archivos de programas de utilidad para la comprobación de errores.
  - Cómo configurar el entorno de desarrollo de aplicaciones, lo que incluye instrucciones específicas para funciones de Java y WebSphere MQ.
  - Cómo configurar la base de datos de ejemplo.
  - Cómo migrar las aplicaciones desde versiones anteriores de DB2.
  - Cómo crear y ejecutar applets, aplicaciones y rutinas de Java.
  - Cómo crear y ejecutar procedimientos de SQL.
  - Cómo crear y ejecutar aplicaciones y rutinas en C/C++.
  - Cómo crear y ejecutar aplicaciones y rutinas en COBOL de IBM y de Micro Focus.
  - Cómo crear y ejecutar aplicaciones de REXX en AIX y Windows.
  - Cómo crear y ejecutar aplicaciones en C# y Visual Basic .NET y rutinas en CLR .NET en Windows.
  - Cómo crear y ejecutar aplicaciones con ActiveX Data Objects (ADO) utilizando Visual Basic y Visual C++ en Windows.
  - Cómo crear y ejecutar aplicaciones con objetos de datos remotos mediante Visual C++ en Windows.

Parte 1. Introducción

Capítulo 1. Visión general de las interfaces de programación soportadas

Herramientas de DB2 Universal Database para desarrollar aplicaciones 3
Visión general de IBM DB2 Development Add-In . . 4
Interfaces de programación soportadas 5
Interfaces de programación que reciben soporte de DB2 5
Interfaces de programación de aplicaciones de DB2 7
SQL incorporado 8
Interfaz de nivel de llamada de DB2 9
CLI de DB2 frente a SQL dinámico incorporado 10
Java Database Connectivity (JDBC) 12
SQL incorporado para Java (SQLJ). 13
Objetos de datos ActiveX y Objetos de datos remotos. 13
DBI Perl 14
Herramientas de usuario final de ODBC. . . . 15
DB2 .NET Data Provider 15
Aplicaciones Web 16
Herramientas para crear aplicaciones Web . . . 16
WebSphere Studio 16
XML Extender 17
Habilitación de MQSeries. 17
Net.Data 18
Funciones de programación 18
Funciones de programación de DB2 18
Procedimientos almacenados de DB2 19
Métodos y funciones definidas por el usuario de DB2 20
Centro de desarrollo 20
Tipos definidos por el usuario (UDT) y objetos grandes (LOB) 22
Rutinas de automatización de OLE 23
Funciones de tabla de OLE DB 23
Activadores de DB2 24

Herramientas de DB2 Universal Database para desarrollar aplicaciones

Puede utilizar distintas herramientas para desarrollar aplicaciones. DB2® Universal Database suministra las siguientes herramientas para ayudarle a escribir y probar las sentencias de SQL en las aplicaciones y para ayudarle a supervisar su rendimiento.

Nota: No todas las herramientas están disponibles en todas las plataformas.

Centro de control

Una interfaz gráfica que muestra objetos de bases de datos (como bases de datos, tablas y paquetes) y sus mutuas relaciones. Utilice el Centro de control para realizar tareas administrativas como configurar el sistema, gestionar directorios, hacer copia de seguridad y recuperar el sistema, planificar trabajos y gestionar soportes.

DB2 también proporciona los siguientes recursos:

Editor de mandatos
Se utiliza para entrar mandatos de DB2 y sentencias de SQL en una ventana interactiva y para ver el resultado de la ejecución en una ventana de resultados. Puede desplazarse por los resultados y guardar la salida en un archivo.

Centro de tareas
Se utiliza para crear scripts, que puede almacenar e invocar posteriormente. Estos scripts pueden contener mandatos de DB2, sentencias de SQL o mandatos del sistema operativo. Puede planificar scripts para que se ejecuten de forma desatendida. Puede ejecutar estos trabajos una vez o puede definirlos de modo que se ejecuten según una planificación repetitiva. Una planificación repetitiva resulta especialmente útil para tareas como copias de seguridad. El Centro de tareas se puede

utilizar también para supervisar el sistema a fin de detectar avisos precoces de problemas potenciales o para automatizar acciones de corrección de problemas.

Diario Se utiliza para ver los siguientes tipos de información: toda la información disponible sobre trabajos cuya ejecución está pendiente, que se están ejecutando o que se han terminado de ejecutar, el registro histórico de recuperación, el registro de alertas y el registro de mensajes. También puede utilizar el Diario para revisar los resultados de trabajos que se ejecutan de forma desatendida.

Valores de herramientas
Se utiliza para cambiar los valores del Centro de tareas.

Supervisor de sucesos
Recopila información sobre el rendimiento de actividades de bases de datos durante un periodo de tiempo. Su información recopilada proporciona un buen resumen de la actividad correspondiente a un determinado suceso de base de datos: por ejemplo, una conexión de base de datos o una sentencia de SQL.

Visual Explain

Visual Explain, una opción instalable correspondiente al Centro de control, es una interfaz gráfica que le permite analizar y ajustar sentencias de SQL, incluida la consulta de planes de acceso elegidos por el optimizador para sentencias de SQL.

Visión general de IBM DB2 Development Add-In

IBM® DB2® Development Add-In es un conjunto de funciones que se integra directamente en el entorno de desarrollo de Microsoft® Visual Studio .NET para trabajar con servidores DB2 y desarrollar rutinas DB2. Con este módulo adicional, puede:

- Iniciar varias herramientas de administración y desarrollo de DB2
- Crear y gestionar proyectos de DB2 en el Explorador de soluciones
- Acceder y gestionar conexiones de datos de DB2 en IBM Explorer
- Crear y modificar scripts de DB2, incluyendo scripts para crear procedimientos almacenados y funciones definidas por el usuario (UDF)

Barra de herramientas de las Herramientas de DB2:

La barra de herramientas de las Herramientas de DB2 le permite iniciar las diferentes herramientas de administración y desarrollo de DB2. Con la barra de herramientas de las Herramientas de DB2, puede iniciar las siguientes herramientas de DB2:

- Centro de Desarrollo
- Centro de control
- Centro de duplicación
- Editor de mandatos
- Centro de tareas
- Centro de salud
- Diario

Tipo de proyecto de DB2:

IBM DB2 Development Add-In presenta una nueva carpeta Proyectos IBM, que incluye un tipo Proyecto de base de datos DB2 para desarrollar scripts de servidor de bases de datos DB2. Con un Proyecto de DB2, puede:

- Añadir scripts de procedimientos almacenados de SQL nuevos o existentes
- Añadir scripts de UDF nuevos o existentes
- Añadir scripts nuevos o existentes basados en plantillas genéricas
- Especificar opciones de configuración de creación incluyendo orden de creación
- Comprobar los archivos de script en el sistema de gestión de control de la fuente de Microsoft Visual Source Safe

Carpeta Conexiones de datos en IBM Explorer:

IBM DB2 Development Add-In amplía el entorno de Visual Studio .NET añadiendo una nueva herramienta denominada IBM Explorer, una ventana acoplable que es similar a Visual Studio .NET Server Explorer. IBM Explorer proporciona a los usuarios de Visual Studio .NET acceso a conexiones de base de datos IBM utilizando la carpeta Conexiones de datos. La carpeta Conexiones de datos en IBM Explorer está específicamente diseñada para las conexiones del proveedor gestionado de DB2. Desde la carpeta Conexiones de datos de IBM Explorer, puede:

- Trabajar con múltiples conexiones de DB2 con nombre que dan soporte a la conexión de tecnología bajo demanda
- Especificar filtros de catálogos de bases de datos y la puesta en antememoria local para un mayor rendimiento y escalabilidad
- Ver propiedades de objetos de servidor, incluyendo tablas, vistas, procedimientos almacenados y funciones
- Recuperar datos de tablas y vistas
- Ejecutar ejecuciones de prueba para procedimientos almacenados y UDF
- Visualizar código fuente para procedimientos almacenados y funciones
- Generar código ADO .NET utilizando arrastrar y descartar

Editor de SQL de DB2:

IBM DB2 Development Add-In también le proporciona el Editor de SQL de DB2. Con el editor, puede cambiar y visualizar el código en las rutinas y scripts de DB2. El Editor de SQL de DB2 incluye las características siguientes:

- Texto en colores para aumentar la legibilidad del SQL.
- Integración con la función Microsoft Visual Studio .NET IntelliSense, que permite la complementación automática inteligente al escribir scripts DB2.

Interfaces de programación soportadas

Las secciones siguientes contienen una visión general de las interfaces de programación soportadas.

Interfaces de programación que reciben soporte de DB2

Puede utilizar varias interfaces de programación diferentes para gestionar bases de datos DB2® o para acceder a las mismas. Puede:

- Utilizar las API de DB2 para realizar funciones administrativas como copia de seguridad y restauración de bases de datos.
- Incorporar sentencias de SQL estático y dinámico en sus aplicaciones.

- Codificar llamadas a funciones de Interfaz de nivel de llamada de DB2 (CLI de DB2) en sus aplicaciones para invocar sentencias de SQL dinámico.
- Desarrollar aplicaciones y applets Java™ que llaman a la interfaz de programación de aplicaciones Java Database Connectivity (API JDBC).
- Desarrollar aplicaciones Visual Basic y Visual C++ de Microsoft® que cumplen las especificaciones DAO (Data Access Object) y RDO (Remote Data Object) y aplicaciones ADO (ActiveX Data Object) que hacen uso de OLE DB Bridge.
- Desarrollar aplicaciones ADO.NET utilizando DB2 .NET Data Provider, OLE DB .NET Data Provider u ODBC .NET Data Provider.
- Desarrollar aplicaciones mediante herramientas de IBM® o de otros proveedores, tales como Net.Data®, Excel, Perl, y herramientas de usuario final de ODBC (Open Database Connectivity), tales como Lotus® Approach y su lenguaje de programación, LotusScript.

El modo en que la aplicación acceda a bases de datos DB2 dependerá del tipo de aplicación que desee desarrollar. Por ejemplo, si desea una aplicación de entrada de datos, es posible que elija incorporar sentencias de SQL estático en la aplicación. Si desea una aplicación que realice consultas en la World Wide Web, es posible que elija Net.Data, Perl o Java.

Aparte del modo en que la aplicación accede a los datos, también debe tener en cuenta lo siguiente:

- Control de valores de datos utilizando:
  - Tipos de datos (integrados o definidos por el usuario)
  - Restricciones de comprobación de tablas
  - Restricciones de integridad referencial
  - Vistas utilizando la opción CHECK
  - Lógica de aplicación y tipos de variable
- Control de las relaciones entre valores de datos utilizando:
  - Restricciones de integridad referencial
  - Activadores
  - Lógica de aplicación
- Ejecución de programas en el servidor utilizando:
  - Procedimientos almacenados
  - Funciones definidas por el usuario
  - Activadores

Observará que esta lista menciona algunas funciones más de una vez, como los activadores. Esto refleja la flexibilidad de estas funciones para ajustarse a más de un criterio de diseño.

La principal y más fundamental decisión es si debe o no mover la lógica para aplicar las normas relacionadas con la aplicación sobre los datos a la base de datos.

La principal ventaja de transferir lógica centrada en los datos de la aplicación a la base de datos es que la aplicación pasa a ser más independiente de los datos. La lógica asociada a los datos se centraliza en un lugar, la base de datos. Esto significa que puede cambiar datos o lógica de datos una vez de forma que ello afecte a *todas* las aplicaciones inmediatamente.

Esta última ventaja es muy potente, pero también debe tener en cuenta que cualquier lógica de datos que se coloque en la base de datos afecta a *todos* los usuarios de los datos por igual. Debe tener en cuenta si las normas y restricciones que desea imponer en los datos se aplican a todos los usuarios de los datos o únicamente a los usuarios de la aplicación.

Los requisitos de la aplicación también pueden afectar a si se deben aplicar las normas en la base de datos o en la aplicación. Por ejemplo, es posible que tenga que procesar errores de validación sobre entrada de datos en un orden específico. En general, debería realizar estos tipos de validación de datos en el código de la aplicación.

También debería tener en cuenta el entorno de cálculo en que se utiliza la aplicación. Debe tener en cuenta la diferencia entre llevar a cabo lógica en las máquinas cliente frente a ejecutar la lógica en las máquinas servidor de bases de datos, generalmente más potentes, utilizando procedimientos almacenados, UDF o una combinación de ambos.

En algunos casos, la respuesta correcta consiste en incluir el cumplimiento tanto en la aplicación (quizás debido a los requisitos específicos de la aplicación) y en la base de datos (quizás debido a otros usuarios interactivos fuera de la aplicación).

Conceptos relacionados:

- "La Interfaz de nivel de llamada (CLI) de DB2 y el SQL dinámico incorporado" en la página 140
- "SQL incorporado" en la página 8
- "Interfaz de nivel de llamada de DB2" en la página 9
- "Interfaces de programación de aplicaciones de DB2" en la página 7
- "Objetos de datos ActiveX y Objetos de datos remotos" en la página 13
- "DBI Perl" en la página 14
- "Herramientas de usuario final de ODBC" en la página 15
- "Herramientas para crear aplicaciones Web" en la página 16
- "Java Database Connectivity (JDBC)" en la página 12

Interfaces de programación de aplicaciones de DB2

Puede que sus aplicaciones tengan que realizar algunas tareas de administración de bases de datos, como crear, activar, hacer copia de seguridad o restaurar una base de datos. DB2® proporciona numerosas API para que pueda realizar estar tareas desde sus aplicaciones, incluidas aplicaciones de SQL incorporado y CLI de DB2. Esto le permite programar en sus aplicaciones las mismas funciones administrativas que puede realizar mediante las herramientas de administración del servidor DB2 disponibles en el Centro de control.

Además, es posible que tenga que llevar a cabo tareas específicas que sólo se pueden realizar mediante las API de DB2. Por ejemplo, puede que desee recuperar el texto de un mensaje de error para que la aplicación lo pueda mostrar al usuario final. Para recuperar el mensaje, debe utilizar la API Obtener mensaje de error.

Conceptos relacionados:

- "Consideraciones sobre autorización para API" en la página 53
- "API administrativas en SQL incorporado o en programas de CLI de DB2" en la página 43

SQL incorporado

Structured Query Language (SQL) es el lenguaje de interfaz de bases de datos que se utiliza para acceder a bases de datos DB2® y para manipular datos de las mismas. Puede incorporar sentencias de SQL en sus aplicaciones, lo que les permitirá realizar cualquier tarea soportada por SQL, como recuperar o almacenar datos. Mediante DB2, puede codificar sus aplicaciones de SQL incorporado en los lenguajes de programación C/C++, COBOL, FORTRAN, Java™ (SQLJ) y REXX.

Nota: Los lenguajes de programación REXX y Fortran no se han mejorado desde la Versión 5 de DB2 Universal Database.

Una aplicación en la que incorpora sentencias de SQL se denomina programa de lenguaje principal. El lenguaje de programación que utilice para crear un programa de lenguaje principal se denomina lenguaje principal. El programa y el lenguaje se definen de este modo porque pueden alojar o acomodar sentencias de SQL.

Para sentencias de SQL estático, sabe antes del momento de la compilación el tipo de sentencia de SQL y los nombres de tablas y columnas. Lo único que no sabe son los valores de datos específicos que la sentencia va a buscar o a actualizar. Puede representar estos valores en variables del lenguaje principal. Debe precompilar, vincular y luego compilar las sentencias de SQL estático antes de ejecutar la aplicación. SQL estático se ejecuta mejor en bases de datos cuyas estadísticas no cambian mucho. De lo contrario, las sentencias quedarán rápidamente obsoletas.

Por el contrario, las sentencias de SQL dinámico son aquellas que la aplicación crea y ejecuta en el tiempo de ejecución. Una aplicación interactiva que solicita al usuario final partes clave de una sentencia de SQL, como los nombres de las tablas y las columnas que hay que buscar, es un buen ejemplo de SQL dinámico. La aplicación crea la sentencia de SQL mientras se está ejecutando y luego somete la sentencia para que se procese.

Puede escribir aplicaciones que tengan sentencias de SQL estático, sentencias de SQL dinámico o una combinación de ambas.

Generalmente, las sentencias de SQL estático resultan ideales para aplicaciones de alto rendimiento con transacciones predefinidas. Un sistema de reservas constituye un buen ejemplo de este tipo de aplicación.

Generalmente, las sentencias de SQL dinámico resultan ideales para aplicaciones que se ejecutan contra una base de datos que cambia con rapidez y en la que las transacciones se tienen que especificar en el tiempo de ejecución. Una interfaz de consulta interactiva es un buen ejemplo de este tipo de aplicación.

Cuando incorpore sentencias de SQL en su aplicación, debe precompilar y vincular la aplicación a una base de datos siguiendo los pasos siguientes:

1. Cree archivos fuente que contengan programas con sentencias de SQL incorporado.
2. Conecte con una base de datos y luego precompile cada archivo fuente.

   El precompilador convierte las sentencias de SQL de cada archivo fuente en llamadas a API en tiempo de ejecución de DB2 al gestor de bases de datos. El precompilador también genera un paquete de acceso en la base de datos y, opcionalmente, un archivo de vinculación, si especifica que desea que se cree uno.

El paquete de acceso contiene planes de acceso seleccionados por el optimizador de DB2 para las sentencias de SQL estático de la aplicación. Los planes de acceso contienen la información que necesita el gestor de bases de datos para ejecutar las sentencias de SQL estático de la forma más eficiente, según determine el optimizador. Para sentencias de SQL dinámico, el optimizador crea planes de acceso cuando el usuario ejecuta la aplicación.

El archivo de vinculación contiene las sentencias de SQL y otros datos necesarios para crear un paquete de acceso. Puede utilizar el archivo de vinculación para revincular la aplicación más adelante sin tener que precompilarla primero. La revinculación crea planes de acceso optimizados para las condiciones actuales de la base de datos. Tiene que revincular la aplicación si va a acceder a una base de datos distinta de aquella contra la cual se precompiló. Debe revincular la aplicación si las estadísticas de la base de datos han cambiado desde la última vinculación.

3. Compile los archivos fuente modificados (y otros archivos sin sentencias de SQL) mediante el compilador del lenguaje principal.
4. Enlace los archivos de objetos con las bibliotecas de DB2 y del lenguaje principal para generar un programa ejecutable.
5. Vincule el archivo de vinculación para crear el paquete de acceso si esto no se ha hecho en el momento de la precompilación o si se va a acceder a una base de datos distinta.
6. Ejecute la aplicación. La aplicación accede a la base de datos utilizando el plan del paquete.

Conceptos relacionados:

- "SQL incorporado en aplicaciones REXX" en la página 531
- "Sentencias de SQL incorporado en C y C++" en la página 150
- "Sentencias de SQL incorporado en COBOL" en la página 196
- "Sentencias de SQL incorporado en FORTRAN" en la página 219
- "SQL incorporado para Java (SQLJ)" en la página 13

Tareas relacionadas:

- "Incorporación de sentencias de SQL en un lenguaje principal" en la página 61

Interfaz de nivel de llamada de DB2

DB2® CLI es una interfaz de programación que las aplicaciones C y C++ pueden utilizar para acceder a bases de datos DB2. CLI de DB2 se basa en la especificación Open Database Connectivity (ODBC) de Microsoft® y en el estándar CLI de ISO. Puesto que CLI de DB2 se basa en estándares de la industria, los programadores de aplicaciones que estén familiarizados con estas interfaces de bases de datos se pueden beneficiar de una curva de aprendizaje más corta.

Cuando utiliza CLI de DB2, la aplicación pasa sentencias de SQL dinámico como argumentos de función al gestor de bases de datos para que las procese. Desde este punto de vista, CLI de DB2 constituye una alternativa a SQL dinámico incorporado.

También se pueden ejecutar sentencias de SQL como SQL estático en una aplicación CLI, ODBC o JDBC. La función Static Profiling de CLI/ODBC/JDBC permite a los usuarios finales de una aplicación sustituir el uso de SQL dinámico por SQL estático en muchos casos.

Puede crear una aplicación ODBC sin utilizar un gestor de controladores ODBC y simplemente utilizar el controlador ODBC de DB2 en cualquier plataforma enlazando la aplicación con `libdb2` en UNIX® y con `db2cli.lib` en sistemas operativos Windows®. Los programas de ejemplo de CLI de DB2 lo demuestran. Estos programas de ejemplo se encuentran en `sqllib/samples/cli` para UNIX, y en `sqllib\samples\cli` para los sistemas operativos Windows.

No necesita precompilar ni vincular las aplicaciones CLI de DB2 porque utilizan paquetes de acceso común que se suministran con DB2. Simplemente tiene que compilar y enlazar la aplicación.

Sin embargo, antes de que las aplicaciones CLI de DB2 u ODBC puedan acceder a bases de datos DB2, los archivos de vinculación CLI de DB2 que vienen con DB2 AD Client se deben vincular a cada una de las bases de datos DB2 a la que se vaya a acceder. Esto se produce automáticamente con la ejecución de la primera sentencia, pero recomendamos que el administrador de bases de datos vincule los archivos de vinculación desde un cliente de cada plataforma que vaya a acceder a una base de datos DB2.

Por ejemplo, supongamos que tiene clientes AIX®, del Entorno operativo Solaris y de Windows 98, cada uno de los cuales accede a dos bases de datos DB2. El administrador debe vincular los archivos de vinculación de un cliente AIX en cada base de datos a la que se vaya a acceder. A continuación, el administrador debe vincular los archivos de vinculación de un cliente del Entorno Operativo Solaris de cada base de datos a la que se accederá. Finalmente, el administrador debe hacer lo mismo en un cliente Windows 98.

Conceptos relacionados:

- "API administrativas en SQL incorporado o en programas de CLI de DB2" en la página 43
- "CLI de DB2 frente a SQL dinámico incorporado" en la página 10

Tareas relacionadas:

- "Creating static SQL with CLI/ODBC/JDBC Static Profiling" en la publicación *CLI Guide and Reference, Volume 1*

CLI de DB2 frente a SQL dinámico incorporado

Puede desarrollar aplicaciones dinámicas mediante sentencias de SQL dinámico incorporado o DB2® CLI. En ambos casos, las sentencias de SQL se preparan y se procesan en el tiempo de ejecución. Cada método tiene ventajas exclusivas.

Las ventajas de CLI de DB2 son las siguientes:

Portabilidad Las aplicaciones CLI de DB2 utilizan un conjunto estándar de funciones para pasar sentencias de SQL a la base de datos. Todo lo que tiene que hacer es compilar y enlazar aplicaciones CLI de DB2 antes de ejecutarlas. Por el contrario, debe precompilar las aplicaciones de SQL incorporado, compilarlas y luego vincularlas a la base de datos antes de ejecutarlas. Este proceso enlaza de forma eficiente la aplicación a una determinada base de datos.

No hay vinculación
No tiene que vincular aplicaciones CLI de DB2 individuales a cada base de datos a la que acceden. Sólo tiene que vincular los archivos que se suministran con CLI de DB2 una vez para todas las

aplicaciones CLI de DB2. Esto permite reducir significativamente la cantidad de tiempo empleada en gestionar las aplicaciones.

Captación y entrada ampliadas
Las funciones CLI de DB2 le permiten recuperar varias filas de la base de datos en una matriz con una sola llamada. También le permiten ejecutar una sentencia de SQL varias veces utilizando una matriz de variables de entrada.

Interfaz coherente ante el catálogo
Los sistemas de bases de datos contienen tablas de catálogo que tienen información sobre la base de datos y sus usuarios. El formato de estos catálogos puede variar entre sistemas. CLI de DB2 proporciona una interfaz coherente para consultar información del catálogo sobre componentes como tablas, columnas, claves principales y foráneas y privilegios de usuarios. Esto protege la aplicación ante cambios en el catálogo entre releases de servidores de bases de datos y ante diferencias entre servidores de bases de datos. No tiene que escribir consultas del catálogo que sean específicas de un determinado servidor o versión de producto.

Conversión de datos ampliada
CLI de DB2 convierte automáticamente datos entre los tipos de datos SQL y C. Por ejemplo, al captar cualquier tipo de datos SQL en un tipo de datos char de C se convierten en una representación de serie de caracteres. Esto hace que CLI de DB2 resulte ideal para aplicaciones de consulta interactiva.

No hay áreas de datos globales
CLI de DB2 elimina la necesidad de disponer de áreas de datos globales controladas por la aplicación y normalmente complejas, como SQLDA y SQLCA, que suelen estar asociadas con aplicaciones de SQL incorporado. En su lugar, CLI de DB2 asigna y control automáticamente las estructuras de datos necesarias y proporciona un descriptor de contexto para que la aplicación haga referencia a las mismas.

Recuperar conjuntos de resultados de procedimientos almacenados
Las aplicaciones CLI de DB2 pueden recuperar varias filas y conjuntos de resultados generados por un procedimiento almacenado que reside en un servidor DB2 Universal Database™, un servidor DB2 para MVS™/ESA (Versión 5 o posterior) o un servidor OS/400® (Versión 5 o posterior). El soporte para la recuperación de varios conjuntos de resultados en OS/400 necesita que se aplique el PTF (Program Temporary Fix) SI01761 al servidor. Póngase en contacto con el administrador del sistema OS/400 para asegurarse de que se ha aplicado este PTF.

Cursores desplazables
CLI de DB2 da soporte a los cursores desplazables del servidor que se pueden utilizar junto con una salida de matriz. Esto resulta útil en aplicaciones GUI que muestran información de base de datos en recuadros desplazables en los que utilizan las teclas Re Pág, Av Pág, Inicio y Fin. Puede declarar un cursor como desplazable y luego avanzar o retroceder por el conjunto de resultados una o más filas. También puede captar filas especificando un desplazamiento a partir de la fila actual, el principio o fin de un conjunto de resultados o una fila específica que haya marcado previamente.

Las ventajas de SQL dinámico incorporado son las siguientes:

Seguridad granular
Todos los usuarios de CLI de DB2 comparten los mismos privilegios. SQL incorporado ofrece la ventaja de una seguridad más granular en la que se garantizan privilegios de ejecución del paquete a usuarios determinados.

Más lenguajes soportados
SQL incorporado da soporte a otros lenguajes, además de C y C++. Esto puede resultar una ventaja si prefiere codificar las aplicaciones en otro lenguaje.

Más coherente con SQL estático
Generalmente, SQL dinámico es más coherente con SQL estático. Si ya sabe cómo programar SQL estático, pasar a SQL dinámico no debe resultar tan difícil como pasar a CLI de DB2.

Conceptos relacionados:

- "La Interfaz de nivel de llamada (CLI) de DB2 y el SQL dinámico incorporado" en la página 140
- "Ventajas de CLI de DB2 sobre el SQL incorporado" en la página 141
- "Cuándo utilizar CLI de DB2 o SQL incorporado" en la página 143

Java Database Connectivity (JDBC)

El soporte de DB2® para Java™ incluye JDBC, una interfaz de SQL dinámico que no depende del proveedor y que permite que la aplicación acceda a datos a través de métodos Java estandarizados. JDBC es similar a la CLI de DB2 en cuanto que el usuario no necesita precompilar ni vincular un programa JDBC. Como estándar independiente del proveedor, las aplicaciones JDBC ofrecen mayor portabilidad. Una aplicación escrita utilizando JDBC sólo utiliza SQL dinámico.

JDBC puede resultar especialmente útil para acceder a bases de datos DB2 a través de Internet. Mediante el lenguaje de programación Java puede desarrollar applets y aplicaciones JDBC que accedan a datos de bases de datos DB2 remotas y los manipulen mediante una conexión de red. También puede crear procedimientos almacenados de JDBC que residan en el servidor, accedan al servidor de bases de datos y devuelvan información a una aplicación cliente remota que llame al procedimiento almacenado.

La API JDBC, que es parecida a la API CLI/ODBC, proporciona un modo estándar de acceder a bases de datos desde código Java. El código Java pasa sentencias de SQL como argumentos de método al controlador JDBC de DB2. El controlador maneja las llamadas a la API JDBC desde el código Java cliente.

La portabilidad de Java le permite ofrecer acceso a DB2 a clientes de distintas plataformas, con el único requisito de disponer de un navegador web habilitado para Java o de un entorno de ejecución Java.

JDBC de tipo 2

Las aplicaciones Java basadas en el controlador JDBC de tipo 2 confían en el cliente DB2 para establecer conexión con DB2. Puede iniciar la aplicación desde el escritorio o desde la línea de mandatos, como cualquier otra aplicación. El controlador JDBC de DB2 maneja las llamadas a la API JDBC desde la aplicación y

utiliza la conexión del cliente para comunicar las peticiones al servidor y para recibir los resultados. No puede crear applets Java mediante el controlador JDBC de tipo 2.

Nota: Se recomienda el controlador JDBC de tipo 2 para WebSphere® Application Servers.

JDBC de tipo 3

Si utiliza el controlador JDBC de tipo 3, sólo puede crear applets Java. Los applets Java no necesitan que el cliente DB2 esté instalado en la máquina cliente. Normalmente, el applet se incorpora en una página web HyperText Markup Language (HTML).

Para ejecutar un applet basado en el controlador JDBC de tipo 3, sólo necesita un navegador web habilitado para Java o un visor de applets en la máquina cliente. Cuando carga la página HTML, el navegador baja el applet Java a la máquina, el cual baja los archivos de clases Java y el controlador JDBC de DB2. Cuando el applet llama a la API JDBC para establecer conexión con DB2, el controlador JDBC establece una conexión de red independiente con la base de datos DB2 a través del servidor de applets JDBC que reside en el servidor Web.

Nota: Se desaprueba el uso del controlador JDBC de tipo 3 en la Versión 8.

JDBC de tipo 4

Puede utilizar el controlador JDBC de tipo 4, que es una novedad de la Versión 8, para crear aplicaciones y applets Java. Para ejecutar una aplicación o un applet basado en el controlador de tipo 4 sólo necesita el archivo `db2jcc.jar`. No se necesita ningún cliente DB2.

Para obtener más información sobre el soporte de JDBC de DB2, visite la siguiente página Web:

`http://www.ibm.com/software/data/db2/udb/ad/v8/java`

SQL incorporado para Java (SQLJ)

El soporte de SQL incorporado de Java™ (SQLJ) de DB2® se suministra mediante DB2 AD Client. Con el soporte de SQLJ de DB2, además del soporte de JDBC de DB2, puede crear y ejecutar procedimientos almacenados, aplicaciones y applets SQLJ. Estos programas contienen SQL estático y utilizan sentencias de SQL que están vinculadas a una base de datos DB2.

Para obtener más información sobre el soporte de SQLJ de DB2, visite la siguiente página Web:

`http://www.ibm.com/software/data/db2/udb/ad/v8/java`

Objetos de datos ActiveX y Objetos de datos remotos

Puede escribir aplicaciones de bases de datos Microsoft® Visual Basic y Microsoft Visual C++ que cumplan con las especificaciones Objeto de acceso a datos (DAO) y Objeto de datos remotos (RDO). DB2® también da soporte a las aplicaciones Objeto de datos ActiveX (ADO) que utilizan el puente entre OLE DB y ODBC de Microsoft.

La especificación Objetos de datos ActiveX (ADO) le permite escribir una aplicación que acceda a datos de un servidor de bases de datos y los manipule a través de un proveedor de OLE DB. Las principales ventajas de ADO son su rápido tiempo de desarrollo, su facilidad de uso y el poco espacio que ocupa en disco.

Los Objetos de datos remotos (RDO) proporcionan un modelo de información para acceder a fuentes de datos remotas a través de ODBC. RDO ofrece un conjunto de objetos que facilitan la conexión a una base de datos, la ejecución de consultas y procedimientos almacenados, la manipulación de resultados y la confirmación de cambios en el servidor. Esta especificación está especialmente diseñada para acceder a fuentes de datos relacionales ODBC remotas y facilita la utilización de ODBC sin complejo código de aplicación.

Para ver ejemplos completos de aplicaciones DB2 que utilizan las especificaciones ADO y RDO, consulte los siguientes directorios:
- Para ejemplos de Objeto de datos ActiveX de Visual Basic, consulte `sqllib\samples\VB\ADO`
- Para ejemplos de Objeto de datos remotos de Visual Basic, consulte `sqllib\samples\VB\RDO`
- Para ejemplos de Servidor de transacciones Microsoft Visual Basic, consulte sqllib\samples\VB\MTS
- Para ejemplos de Objeto de datos ActiveX de Visual C++, consulte `sqllib\samples\VC\ADO`

Tareas relacionadas:

- "Creación de aplicaciones ADO con Visual Basic" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de aplicaciones RDO con Visual Basic" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de aplicaciones ADO con Visual C++" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Información relacionada:

- "Ejemplos de Visual Basic" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Ejemplos de Visual C++" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

DBI Perl

DB2® da soporte a la especificación Interfaz de bases de datos (DBI) Perl para el acceso a datos a través del controlador DBD::DB2. El sitio web de DBI Perl de DB2 Universal Database™ se encuentra en:

```
http://www.ibm.com/software/data/db2/perl/
```

y contiene el último controlador DBD::DB2 e información relacionada.

Perl es un lenguaje interpretado y el Módulo DBI de Perl utiliza SQL dinámico. Esto convierte Perl en un lenguaje ideal para crear y revisar con rapidez prototipos de aplicaciones DB2. El Módulo DBI de Perl utiliza una interfaz que es bastante parecida a las interfaces CLI y JDBC. Esto facilita el transporte de prototipos Perl a CLI y JDBC.

Conceptos relacionados:

- "Consideraciones sobre la programación en Perl" en la página 525

Herramientas de usuario final de ODBC

Puede utilizar herramientas de usuario final de ODBC como Lotus® Approach, Microsoft® Access y Microsoft Visual Basic para crear aplicaciones. Las herramientas ODBC ofrecen una alternativa más sencilla para desarrollar aplicaciones que utilizar un lenguaje de programación de alto nivel.

Lotus Approach proporciona dos métodos de acceder a datos DB2®. Puede utilizar la interfaz gráfica para realizar consultas, desarrollar informes y analizar datos, o bien puede desarrollar aplicaciones utilizando LotusScript, un lenguaje de programación con todas sus funciones y orientado a objetos que viene con una amplia matriz de objetos, sucesos, métodos y propiedades, junto con un editor de programas incorporado.

DB2 .NET Data Provider

DB2® .NET Data Provider amplía el soporte de DB2 para la interfaz ADO.NET. DB2 .NET Data Provider proporciona un acceso seguro y de alto rendimiento a datos DB2.

DB2 .NET Data Provider le permite que sus aplicaciones .NET accedan a los siguientes sistemas de gestión de bases de datos:

- DB2 Universal Database™ Versión 8 para sistemas Windows®, UNIX® y Linux
- DB2 Universal Database Versión 6 (o posterior) para OS/390® y z/OS™, a través de DB2 Connect™
- DB2 Universal Database Versión 5, Release 1 (o posterior) para AS/400® e iSeries™, a través de DB2 Connect
- DB2 Universal Database Versión 7.3 (o posterior) para VSE y VM, a través de DB2 Connect

Para desarrollar y ejecutar aplicaciones que hacen uso de DB2 .NET Data Provider, es necesario .NET Framework, Versión 1.0 o 1.1.

Además de DB2 .NET Data Provider, también hay un grupo de complementos a Microsoft® Visual Studio .NET IDE. Estos módulos simplifican la creación de aplicaciones DB2 que hacen uso de la API ADO.NET. También puede utilizar estos módulos para desarrollar objetos del servidor, tales como procedimientos almacenados de SQL y funciones definidas por el usuario.

Las aplicaciones de ejemplo de VB.NET y C#.NET que muestran DB2 .NET Data Provider están disponibles en:

```
http://www.ibm.com/software/data/db2/udb/ad/v8/samples.html
```

Aplicaciones Web

Las secciones siguientes describen los productos y funciones disponibles para crear aplicaciones Web.

Herramientas para crear aplicaciones Web

DB2® Universal Database da soporte a todos los estándares clave de Internet, lo que la convierte en una base de datos ideal para utilizar en la Web. Tiene velocidad en memoria para facilitar las búsquedas en Internet y la comparación compleja de texto combinada con las características de escalabilidad y disponibilidad de una base de datos relacional. Debido a que DB2 Universal Database da soporte a WebSphere®, Java™ y XML Extender, facilita el despliegue de aplicaciones de comercio electrónico.

DB2 Universal Developer's Edition tiene varias herramientas que proporcionan soporte de habilitación de la Web. WebSphere Studio Application Developer, Versión 4, es un entorno de desarrollo integrado (IDE) que le permite crear, probar y desplegar aplicaciones Java en un servidor de aplicaciones WebSphere y en DB2 Universal Database. WebSphere Studio es un grupo de herramientas que combina todos los aspectos relacionados con el desarrollo de sitios Web en una interfaz común. WebSphere Application Server Advanced Edition (un solo servidor) proporciona un potente entorno de despliegue para aplicaciones e-business. Sus componentes le permiten crear y desplegar contenido Web dinámico y personalizado de forma rápida y sencilla.

Conceptos relacionados:

- "WebSphere Studio" en la página 16
- "XML Extender" en la página 17

WebSphere Studio

WebSphere® Studio es un grupo de herramientas que combinan todos los aspectos del desarrollo de sitios Web en una interfaz común. WebSphere Studio facilita más que nunca la creación, ensamblaje, publicación y mantenimiento en cooperación de aplicaciones Web interactivas. Studio consta de los componentes Workbench, Page Designer y Remote Debugger y asistentes y viene con copias de prueba de productos de desarrollo de Web complementarios, como Macromedia Flash, Fireworks, Freehand y Director. WebSphere Studio le permite realizar todo lo que necesita para crear sitios Web interactivos que den soporte a funciones empresariales avanzadas.

WebSphere Application Server Standard Edition (que se suministra con DB2® Universal Developer's Edition) es un componente de WebSphere Studio. Combina la portabilidad de las aplicaciones empresariales del servidor con el rendimiento y capacidad de gestión de tecnologías Java™ para ofrecer una amplia plataforma para diseñar aplicaciones Web basadas en Java. Permite potentes interacciones con bases de datos empresariales y sistemas de transacciones. Puede ejecutar el servidor DB2 en la misma máquina que WebSphere Application Server o en otro servidor Web.

WebSphere Application Server Advanced Edition (que no se suministra con DB2 Universal Developer's Edition) proporciona soporte adicional para aplicaciones Enterprise JavaBean. DB2 Universal Database™ se suministra con WebSphere Application Server Advanced Edition para que se utilice como el repositorio del

servidor administrativo. Incorpora las funciones de servidor para aplicaciones creadas según la especificación EJB de Sun Microsystems, que proporciona soporte para la integración de aplicaciones Web en sistemas empresariales que no son Web.

Conceptos relacionados:

- "Enterprise Java Beans" en la página 520

Información relacionada:

- "Ejemplos de Java WebSphere" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

XML Extender

Extensible Markup Language (XML) es la técnica estándar aceptada para el intercambio de datos entre aplicaciones. Un documento XML es un documento codificado que las personas pueden leer. El texto consta de datos de carácter y códigos de marcación. Los códigos de marcación los define el autor del documento. Se utiliza una Definición de tipo de documento (DTD) para declarar las definiciones de marcación y las restricciones. DB2® XML Extender (que se suministra con DB2 Universal Developer's Edition y con Personal Developer's Edition en Windows®) proporciona un mecanismo mediante el cual los programas pueden manejar datos XML mediante extensiones de SQL.

DB2 XML Extender incorpora tres nuevos tipos de datos: `XMLVARCHAR`, `XMLCLOB` y `XMLFILE`. Extender proporciona funciones definidas por el usuario (UDF) para almacenar, extraer y actualizar documentos XML contenidos dentro de una sola columna o en varias columnas y tablas. La búsqueda se puede realizar en el documento XML entero o se puede basar en componentes estructurales utilizando la vía de acceso de ubicación, que utiliza un subconjunto de Extensible Stylesheet Language Transformation (XSLT) y XPath para XML Path Language.

Para facilitar el almacenamiento de documentos XML como un grupo de columnas, DB2 XML Extender proporciona una herramienta de administración para ayudar al diseñador con la correlación entre XML y base de datos relacional. La Definición de acceso a documento (DAD) se utiliza para mantener los datos estructurales y de correlación correspondientes a los documentos XML. La DAD se define y se almacena como un documento XML, el cual facilita la manipulación y comprensión. Dispone de nuevos procedimientos almacenados para componer o descomponer el documento.

Para obtener más información sobre DB2 XML Extender, visite:

`http://www.ibm.com/software/data/db2/extenders/xmlext/index.html`

Habilitación de MQSeries

Con DB2® Universal Database se suministra un grupo de funciones de MQSeries® que permiten a las aplicaciones DB2 interactuar con operaciones asíncronas de gestión de mensajes. Esto significa que el soporte de MQSeries está disponible para las aplicaciones en cualquier lenguaje de programación que reciba soporte de DB2.

En una configuración básica, un servidor MQSeries está situado en la máquina servidor de bases de datos junto con DB2 Universal Database™. Las funciones de MQSeries están disponibles desde un servidor DB2 y proporcionan acceso a otras aplicaciones MQSeries. Varios clientes DB2 pueden acceder simultáneamente a las

funciones de MQSeries a través de la base de datos. Las operaciones de MQSeries permiten a las aplicaciones DB2 comunicarse de forma síncrona con otras aplicaciones MQSeries. Por ejemplo, las nuevas funciones proporcionan un método sencillo que permite a una aplicación DB2 publicar sucesos de bases de datos en aplicaciones MQSeries remotas, iniciar un flujo de trabajo a través del producto opcional MQSeries Workflow o comunicarse con un paquete de aplicación existente con el producto opcional MQSeries Integrator.

Net.Data

Net.Data® permite el acceso de Internet e intranet a datos de DB2® a través de aplicaciones web. Aprovecha las interfaces (API) del servidor Web, proporcionando un rendimiento superior que las aplicaciones Common Gateway Interface (CGI). Net.Data da soporte al proceso del cliente así como al proceso del servidor con lenguajes tales como Java™, REXX, Perl y C++. Net.Data proporciona lógica adicional y un potente lenguaje de macros. También proporciona soporte de XML, lo que le permite generar códigos XML como salida de su macro Net.Data en lugar de tener que entrar los códigos de forma manual. También puede especificar una hoja de estilo XML (XSL) que se utilizará para formatear y visualizar la salida generada. Net.Data sólo está disponible si se baja de la Web. Para obtener más información, consulte el siguiente sitio Web:

```
http://www-4.ibm.com/software/data/net.data/support/index.html
```

Nota: El soporte de Net.Data se ha estabilizado en DB2 Versión 7.2 y no está previsto realizar mejoras para el soporte de Net.Data en el futuro.

Conceptos relacionados:

- "Herramientas para crear aplicaciones Web" en la página 16
- "XML Extender" en la página 17

Funciones de programación

Las secciones siguientes describen las funciones de programación disponibles con DB2.

Funciones de programación de DB2

DB2® viene con diversas funciones que se ejecutan en el servidor y que puede utilizar para complementar o ampliar sus aplicaciones. Si utiliza las funciones de DB2, no tiene que escribir su propio código para llevar a cabo las mismas tareas. DB2 también le permite almacenar algunas partes de su código en el servidor en lugar de conservar todo el código en las aplicaciones cliente. Esto puede aportar ventajas en cuanto a rendimiento y mantenimiento.

Hay funciones para proteger los datos y para definir relaciones entre los datos. Asimismo, hay funciones relacionales de objetos para crear aplicaciones flexibles y avanzadas. Puede utilizar algunas funciones de más de una forma. Por ejemplo, las restricciones le permite proteger datos y definir relaciones entre valores de datos. Estas son algunas de las funciones clave de DB2:

- Restricciones
- Tipos definidos por el usuario (UDT) y objetos grandes (LOB)
- Funciones definidas por el usuario (UDF)
- Activadores

- Procedimientos almacenados

Para decidir si se deben utilizar o no funciones de DB2, tenga en cuenta los puntos siguientes:

Independencia de la aplicación
Puede hacer que la aplicación sea independiente de los datos que procesa. Mediante funciones de DB2 que se ejecutan en la base de datos puede mantener y cambiar la lógica relacionada con los datos sin que ello afecte a la aplicación. Si tiene que realizar un cambio en dicha lógica, sólo tiene que hacerlo en un lugar, el servidor, y no en cada aplicación que accede a los datos.

Rendimiento
Puede aumentar el rendimiento de la aplicación almacenando y ejecutando partes de la misma en el servidor. Esto envía parte del proceso a máquinas servidor generalmente más potentes y permite reducir el tráfico de la red entre la aplicación cliente y el servidor.

Requisitos de la aplicación
Es posible que la aplicación tenga lógica exclusiva que otras aplicaciones no tengan. Por ejemplo, si la aplicación procesa errores de entrada de datos en un orden determinado que no resultaría adecuado para otras aplicaciones, probablemente deseará escribir su propio código para manejar esta situación.

En algunos casos, puede decidir utilizar funciones de DB2 que se ejecuten en el servidor porque así las pueden utilizar varias aplicaciones. En otros casos, deseará conservar la lógica en la aplicación porque sólo esta la utiliza.

Conceptos relacionados:

- "Procedimientos almacenados de DB2" en la página 19
- "Métodos y funciones definidas por el usuario de DB2" en la página 20
- "Tipos definidos por el usuario (UDT) y objetos grandes (LOB)" en la página 22
- "Activadores de DB2" en la página 24

Procedimientos almacenados de DB2

Normalmente, las aplicaciones acceden a la base de datos a través de la red. Esto puede dar lugar a un bajo rendimiento si se tienen que devolver muchos datos. Un procedimiento almacenado se ejecuta en el servidor de bases de datos. Una aplicación cliente puede llamar al procedimiento almacenado, el cual lleva a cabo el acceso a la base de datos sin devolver datos innecesarios a través de la red. El procedimiento almacenado sólo tiene que devolver los resultados que necesita la aplicación cliente.

Puede obtener varias ventajas de la utilización de procedimientos almacenados:

Reducción del tráfico de la red
Si agrupa sentencias de SQL puede reducir el tráfico de la red. Una aplicación típica necesita dos viajes por la red para cada sentencia de SQL. La agrupación de sentencias de SQL da lugar a dos viajes a través de la red para cada grupo de sentencias, lo que mejora el rendimiento de las aplicaciones.

Acceso a funciones que sólo existen en el servidor
Los procedimientos almacenados pueden acceder a mandatos que

sólo se ejecutan en el servidor, como LIST DATABASE DIRECTORY y LIST NODE DIRECTORY; pueden ofrecer las ventajas de mayor memoria y espacio de disco de las máquinas servidor, y pueden acceder a cualquier software adicional instalado en el servidor.

Cumplimiento de las normas empresariales
Puede utilizar procedimientos almacenados para definir normas empresariales comunes a varias aplicaciones. Es otra forma de definir normas empresariales, además de utilizar restricciones y activadores.

Cuando una aplicación llama al procedimiento almacenado, procesa los datos de forma coherente de acuerdo a las normas definidas en el procedimiento almacenado. Si tiene que cambiar las normas, sólo tiene que realizar el cambio una vez en el procedimiento almacenado, no en cada aplicación que lo llama.

Conceptos relacionados:

• "Centro de desarrollo" en la página 20

Métodos y funciones definidas por el usuario de DB2

Es posible que las funciones integradas que se suministran con SQL no cumplan con todos los requisitos de sus aplicaciones. Para permitirle ampliar estas funciones, DB2® da soporte a funciones definidas por el usuario (UDF) y métodos. Puede escribir su propio código en Visual Basic, C/C++, Java™ o SQL para realizar operaciones dentro de cualquier sentencia de SQL que devuelva un solo valor escalar o una tabla.

Las UDF y los métodos le ofrecen una gran flexibilidad. Devuelven un solo valor escalar como parte de una expresión. Además, las funciones pueden devolver tablas enteras a partir de fuentes que no sean de base de datos, como hojas de cálculo.

Las UDF y los métodos proporcionan un método de estandarizar las aplicaciones. Al implantar un conjunto común de rutinas, muchas aplicaciones pueden procesar datos del mismo modo, asegurando así resultados coherentes.

Las funciones definidas por el usuario y los métodos también dan soporte a la programación orientada a objetos en las aplicaciones. Permiten realizar abstracciones, lo que le permite definir interfaces comunes que se pueden utilizar para llevar a cabo operaciones en objetos de datos. Además, permiten realizar encapsulaciones, lo que le permite controlar el acceso a los datos subyacentes de un objeto y protegerlos frente a una manipulación directa y posibles daños.

Centro de desarrollo

El Centro de desarrollo de DB2® proporciona un entorno de desarrollo de fácil utilización para crear, instalar y probar procedimientos almacenados. Le permite concentrarse en la creación de la lógica del procedimiento almacenado en vez de hacerlo en los detalles del registro, la creación y la instalación de procedimientos almacenados en un servidor DB2. Además, con el Centro de desarrollo puede desarrollar procedimientos almacenados en un sistema operativo y crearlos en otros sistemas operativos de servidor.

El Centro de desarrollo es una aplicación gráfica que soporta un rápido desarrollo. Mediante el Centro de desarrollo puede realizar las tareas siguientes:

- Crear procedimientos almacenados nuevos.
- Crear procedimientos almacenados en servidores DB2 locales y remotos.
- Modificar y volver a crear procedimientos almacenados existentes.
- Probar y depurar la ejecución de procedimientos almacenados instalados.

Puede activar el Centro de desarrollo como una aplicación separada del grupo de programas DB2 Universal Database™ o lo puede activar desde cualquiera de las aplicaciones de desarrollo siguientes:

- Microsoft® Visual Studio
- Microsoft Visual Basic
- IBM® VisualAge® para Java™

También puede iniciar el Centro de desarrollo desde el Centro de control para DB2 para OS/390®. Puede iniciar el Centro de desarrollo como un proceso separado desde el menú Herramientas del Centro de control, desde la barra de herramientas o desde la carpeta Procedimientos almacenados. Además, desde la ventana Proyecto del Centro de desarrollo puede exportar uno o más procedimientos almacenados SQL creados para un servidor DB2 para OS/390 a un archivo específico que se puede ejecutar en el procesador de línea de mandatos (CLP).

El Centro de desarrollo gestiona el trabajo utilizando proyectos. Cada proyecto del Centro de desarrollo guarda las conexiones con bases de datos específicas, como por ejemplo servidores DB2 para OS/390. Además, puede crear filtros para visualizar subconjuntos de los procedimientos almacenados en cada una de las bases de datos. Cuando abra un proyecto nuevo o existente del Centro de desarrollo, puede filtrar los procedimientos almacenados de forma que los visualice en base a su nombre, esquema, lenguaje o ID de colección (sólo para OS/390).

En el proyecto del Centro de desarrollo se guarda información de conexión; por consiguiente, cuando abra un proyecto existente, automáticamente se le solicitará que entre su ID de usuario y contraseña para la base de datos. Utilizando el asistente para Insertar procedimiento almacenado SQL, puede crear procedimientos almacenados SQL en un servidor DB2 para OS/390. Para un procedimiento almacenado SQL creado para un servidor DB2 para OS/390, puede establecer opciones específicas de compilación, previnculación, enlace, vinculación, ejecución, entorno WLM y seguridad externa.

Asimismo, puede obtener de SQL información de costes sobre el procedimiento almacenado SQL, la cual incluye información sobre el tiempo de CPU y otras informaciones de costes para la hebra en la que se ejecuta el procedimiento almacenado SQL. En concreto, puede obtener información de costes sobre el tiempo de espera de contención de enclavamiento/bloqueo, sobre el número de obtenciones de página, el número de E/S de lectura y el número de E/S de grabación.

Para obtener información de costes, el Centro de desarrollo conecta con un servidor DB2 para OS/390, ejecuta la sentencia de SQL y llama a un procedimiento almacenado (DSNWSPM) para averiguar cuánto tiempo de CPU ha utilizado el procedimiento almacenado SQL.

Conceptos relacionados:

- “Procedimientos almacenados de DB2” en la página 19

• "Rutinas de automatización de OLE" en la página 23

Tipos definidos por el usuario (UDT) y objetos grandes (LOB)

Cada elemento de datos de la base de datos se almacena en una columna de una tabla y cada columna se define con un tipo de datos. El tipo de datos impone límites en los tipos de valores que puede colocar en la columna y en las operaciones que puede realizar en los mismos. Por ejemplo, una columna de enteros sólo puede contener números comprendidos dentro de un rango fijo. DB2® incluye un conjunto de tipos de datos integrados con características y comportamientos definidos: series de caracteres, numéricos, valores de fecha y hora, objetos grandes, valores nulos, series gráficas, series binarias y enlaces de datos.

Sin embargo, es posible que en algunas ocasiones los tipos de datos integrados no cumplan con los requisitos de las aplicaciones. DB2 proporciona tipos definidos por el usuario (UDT) que le permiten definir tipos de datos diferenciados que necesite para las aplicaciones.

Los UDT se basan en los tipos de datos integrados. Cuando define un UDT, también define las operaciones que son válidas para el UDT. Por ejemplo, puede definir un tipo de datos MONEY basado en el tipo de datos DECIMAL. Sin embargo, para el tipo de datos MONEY puede permitir únicamente operaciones de suma y resta, pero no operaciones de multiplicación y división.

Los objetos grandes (LOB) le permiten almacenar y manipular objetos de datos grandes y complejos en la base de datos; objetos como audio, vídeo, imágenes y documentos grandes.

La combinación de UDT y LOB le ofrece una potencia adicional. Ya no está limitado a utilizar los tipos de datos integrados que se suministran con DB2 para perfilar datos empresariales y para capturar la semántica de dichos datos. Puede utilizar UDT para definir estructuras de datos grandes y complejas para aplicaciones avanzadas.

Además de ampliar los tipos de datos integrados, los UDT proporcionan otras ventajas:

Soporte de programación orientada a objetos en las aplicaciones
Puede agrupar objetos parecidos en tipos de datos relacionados. Estos tipos tienen un nombre, una representación interna y un comportamiento específico. Mediante la utilización de UDT, puede indicar a DB2 el nombre del nuevo tipo y el modo en que se debe representar internamente. Un LOB es una de las posibles representaciones internas para su nuevo tipo y es la representación más adecuada para estructuras de datos grandes y complejas.

Integridad de los datos a través de una estricta tipificación y encapsulación
La tipificación estricta garantiza que sólo las funciones y operaciones definidas en el tipo diferenciado se pueden aplicar al tipo. La encapsulación asegura que el comportamiento de los UDT está restringido por las funciones y operadores que se pueden aplicar a los mismos. En DB2, el comportamiento correspondiente a los UDT se puede proporcionar en forma de funciones definidas por el usuario (UDF), que se pueden escribir de modo que se acomoden a una amplia gama de requisitos del usuario.

Rendimiento a través de la integración en el gestor de bases de datos
Puesto que los UDT se representan internamente, igual que los tipos de datos integrados, comparten el mismo código eficiente que los tipos de datos integrados para implantar funciones integradas, operadores de comparación, índices y otras funciones. La excepción a esto son los UDT que utilizan LOB, los cuales no se pueden utilizar con operadores de comparación ni índices.

Conceptos relacionados:

- "Uso de objetos grandes" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*
- "Tipos definidos por el usuario" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*

Rutinas de automatización de OLE

La automatización de OLE (Object Linking and Embedding) forma parte de la arquitectura OLE 2.0 de Microsoft® Corporation. Con la automatización de OLE, las aplicaciones, independientemente del lenguaje en el que están escritas, pueden exponer sus propiedades y métodos en objetos de automatización de OLE. Otras aplicaciones, como Lotus® Notes o Microsoft Exchange, pueden integrar estos objetos aprovechando estas propiedades y métodos a través de la automatización de OLE.

DB2® para sistemas operativos Windows® proporciona acceso a objetos de automatización de OLE mediante UDF, métodos y procedimientos almacenados. Para acceder a objetos de automatización de OLE e invocar sus métodos, debe registrar los métodos de los objetos como rutinas (UDF, métodos o procedimientos almacenados) en la base de datos. Las aplicaciones DB2 pueden utilizar los métodos invocando las rutinas.

Por ejemplo, puede desarrollar una aplicación que consulte datos de una hoja de cálculo creada mediante un producto como Microsoft Excel. Para ello, debe desarrollar una función de tabla de automatización de OLE que recupere datos de la hoja de trabajo y los devuelva a DB2. Luego DB2 puede procesar los datos, realizar un proceso analítico en línea (OLAP) y devolver el resultado de la consulta a la aplicación.

Conceptos relacionados:

- "Procedimientos almacenados de DB2" en la página 19
- "Centro de desarrollo" en la página 20

Funciones de tabla de OLE DB

Microsoft® OLE DB es un conjunto de interfaces OLE/COM que proporcionan a las aplicaciones acceso uniforme a datos almacenados en distintas fuentes de información. DB2® Universal Database simplifica la creación de aplicaciones OLE DB al permitirle definir funciones de tabla que acceden a una fuente de datos OLE DB. Puede llevar a cabo operaciones que incluyen GROUP BY, JOIN y UNION, en fuentes de datos que exponen sus datos a través de interfaces OLE DB. Por ejemplo, puede definir una función de tabla OLE DB que devuelva una tabla de una base de datos Microsoft Access o de un listín Microsoft Exchange y luego crear un informe que combine datos procedentes de esta función de tabla OLE DB con datos de su base de datos DB2.

La utilización de funciones de tabla OLE DB reduce el esfuerzo que hay que dedicar al desarrollo de aplicaciones al proporcionar acceso integrado a cualquier proveedor de OLE DB. Para funciones de tabla de automatización de C, Java™ y OLE, el desarrollador debe implementar la función de tabla, mientras que en el caso de funciones de tabla de OLE DB, un consumidor genérico integrado OLE DB interacciona con cualquier proveedor de OLE DB para recuperar datos. Sólo necesita registrar una función de tabla cuyo tipo de lenguaje sea OLEDB y especificar el proveedor de OLE DB y el conjunto de filas apropiado como fuente de datos. No es necesario que realice ninguna programación de UDF para aprovechar las funciones de tabla de OLE DB.

Conceptos relacionados:

- "Objetivo de IBM OLE DB Provider para DB2" en la página 241
- "Habilitación automática de servicios OLE DB por parte de IBM OLE DB Provider" en la página 245

Información relacionada:

- "Soporte de IBM OLE DB Provider de interfaces y componentes de OLE DB" en la página 250
- "Soporte de IBM OLE DB Provider para propiedades de OLE DB" en la página 253

Activadores de DB2

Un activador define un conjunto de acciones que se ejecutan en respuesta a una operación de inserción (INSERT), actualización (UPDATE) o supresión (DELETE) sobre una tabla especificada. Cuando se ejecuta dicha operación de SQL, se dice que el activador se activa. El activador se puede activar antes de la operación de SQL o después de ella. Puede definir un activador mediante la sentencia CREATE TRIGGER de SQL.

Puede utilizar activadores que se ejecutan antes de una actualización o inserción con varias finalidades:

- Para comprobar o modificar valores antes de que se actualicen o inserten realmente en la base de datos. Esto resulta útil si tiene que transformar datos desde el modo que el usuario los ve a otro formato interno de la base de datos.
- Para ejecutar otras operaciones que no son de base de datos codificadas en funciones definidas por el usuario.

Similarmente, puede utilizar activadores que se ejecutan después de una actualización o inserción con varias finalidades:

- Para actualizar datos de otras tablas. Esta función resulta útil para mantener relaciones entre datos o para conservar información de seguimiento de auditoría.
- Para comparar con otros datos de la tabla o de otras tablas. Esta función resulta útil para asegurar la integridad de los datos cuando no resulta adecuado utilizar restricciones de integridad referencial o cuando las restricciones de comprobación de tabla limitan la comprobación únicamente a la tabla actual.
- Para ejecutar operaciones que no son de base de datos codificadas en funciones definidas por el usuario. Esta función resulta útil para emitir alertas o para actualizar información externa a la base de datos.

La utilización de activadores presenta varias ventajas:

Desarrollo más rápido de aplicaciones

Los activadores se almacenan en la base de datos y están disponibles para todas las aplicaciones, lo que le releva de la necesidad de codificar funciones equivalentes para cada aplicación.

Cumplimiento global de las normas empresariales

Los activadores se definen una vez y los utilizan todas las aplicaciones que utilizan los datos controlados por los activadores.

Facilidad de mantenimiento

Los cambios sólo se tienen que hacer una vez en la base de datos, en lugar de tener que hacerlos en cada aplicación que utiliza un activador.

Conceptos relacionados:

- "Activadores en el desarrollo de aplicaciones" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*
- "Directrices para la creación de activadores" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*

Capítulo 2. Codificación de una aplicación DB2

Requisitos previos para programación 27
Visión general de la codificación de aplicaciones DB2 28
Programación de una aplicación autónoma . . . 28
Creación de una sección de declaración de una aplicación autónoma 29
Declaración de variables que interactúan con el gestor de bases de datos 29
Declaración de variables que representan objetos de SQL 30
Declaración de variables del lenguaje principal con el Generador de declaraciones db2dclgn . . 32
Relación de variables del lenguaje principal con una sentencia de SQL 33
Declaración de la SQLCA para el manejo de errores 34
Manejo de errores utilizando la sentencia WHENEVER 35
Adición de sentencias no ejecutables a una aplicación 36
Conexión de una aplicación con una base de datos 36
Codificación de transacciones 37
Finalización de una transacción con la sentencia COMMIT 38
Finalización de una transacción con la sentencia ROLLBACK 39
Finalización de un programa de aplicación . . . 40
Finalización implícita de una transacción en una aplicación autónoma 40
Infraestructura de seudocódigo de aplicación . . 41
Recursos para crear prototipos de sentencias de SQL 42
API administrativas en SQL incorporado o en programas de CLI de DB2 43
Control de valores de datos y relaciones 44
Control de valores de datos 44
Control de valores de datos mediante tipos de datos 44
Control de valores de datos mediante restricciones exclusivas 45
Control de valores de datos mediante restricciones de comprobación de tabla 45
Control de valores de datos mediante restricciones de integridad referencial. 45
Control de valores de datos mediante vistas con la opción Check 46
Control de valores de datos mediante lógica de aplicación y tipos de variables de programa . . 46
Control de relaciones de datos 46
Control de relaciones de datos mediante restricciones de integridad referencial. 47
Control de relaciones de datos mediante activadores 47
Control de relaciones de datos mediante activadores anteriores 48
Control de relaciones de datos mediante activadores posteriores 48
Control de relaciones de datos mediante lógica de aplicación 49
Lógica de aplicación en el servidor 49
Consideraciones sobre autorización para SQL y API 50
Consideraciones sobre autorización para SQL incorporado 50
Consideraciones sobre autorización para SQL dinámico 51
Consideraciones sobre autorización para SQL estático 52
Consideraciones sobre autorización para API . . 53
Prueba de la aplicación 53
Configuración del entorno de prueba para una aplicación 53
Configuración de un entorno de prueba . . . 53
Creación de tablas y vistas de prueba. . . . 54
Generación de datos de prueba 55
Depuración y optimización de una aplicación . . 57

Requisitos previos para programación

Antes de desarrollar una aplicación, necesita el entorno operativo adecuado. Se debe instalar y configurar adecuadamente lo siguiente:

- Un compilador o intérprete soportado para desarrollar las aplicaciones.
- DB2 Universal Database, de forma local o remota.
- DB2 Application Development Client.

Puede desarrollar aplicaciones en un servidor o en cualquier cliente que tenga instalado DB2 Application Development Client. Puede ejecutar aplicaciones con el servidor, DB2 Run-Time Client o DB2 Administrative Client. También puede desarrollar programas Java™ JDBC en uno de estos clientes, suponiendo que instale el componente "Java Enablement" cuando instale el cliente. Esto significa que

puede ejecutar cualquier aplicación DB2 en estos clientes. Sin embargo, a no ser que también instale DB2 Application Development Client con estos clientes, sólo puede desarrollar aplicaciones JDBC en los mismos.

DB2® da soporte a los lenguajes de programación C, C++, Java (SQLJ), COBOL y FORTRAN a través de sus precompiladores. Además, DB2 proporciona soporte para los lenguajes interpretados dinámicamente Perl, Java (JDBC) y REXX.

Nota: El soporte de FORTRAN y REXX se estabilizó en DB2 Versión 5 y no se ha planificado ninguna mejora para el soporte de FORTRAN o REXX en el futuro.

DB2 proporciona una base de datos de ejemplo, que necesitará para ejecutar los programas de ejemplo suministrados.

Tareas relacionadas:

- "Configuración del entorno de desarrollo de aplicaciones" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Configuración de la base de datos de ejemplo" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Visión general de la codificación de aplicaciones DB2

Las secciones siguientes contienen una visión general sobre cómo codificar una aplicación de DB2.

Programación de una aplicación autónoma

Una aplicación autónoma es una aplicación que no llama a objetos de base de datos, como procedimientos almacenados, cuando se ejecuta. Cuando escriba la aplicación, debe asegurarse de que determinadas sentencias de SQL aparezcan al principio y al final del programa para manejar la transición de lenguaje principal a las sentencias de SQL incorporado.

Procedimiento:

Para programar una aplicación autónoma, debe asegurarse de:

1. Crear la sección de declaración.
2. Establecer conexión con la base de datos.
3. Escribir una o más transacciones.
4. Finalizar cada transacción utilizando uno de los siguientes métodos:
   - Confirmar los cambios realizados por la aplicación en la base de datos.
   - Retrotraer los cambios realizados por la aplicación en la base de datos.
5. Finalizar el programa.

Conceptos relacionados:

- "Requisitos previos para programación" en la página 27
- "Infraestructura de seudocódigo de aplicación" en la página 41
- "Recursos para crear prototipos de sentencias de SQL" en la página 42
- "Archivos de ejemplo" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Tareas relacionadas:

- "Creación de una sección de declaración de una aplicación autónoma" en la página 29
- "Conexión de una aplicación con una base de datos" en la página 36
- "Codificación de transacciones" en la página 37
- "Finalización de una transacción con la sentencia COMMIT" en la página 38
- "Finalización de una transacción con la sentencia ROLLBACK" en la página 39
- "Finalización de un programa de aplicación" en la página 40
- "Configuración de un entorno de prueba" en la página 53

Creación de una sección de declaración de una aplicación autónoma

El principio de cada programa debe contener una sección de declaración, que contiene:

- Declaraciones de todas las variables y estructuras de datos que el gestor de bases de datos utiliza para interactuar con el programa de lenguaje principal
- Sentencias de SQL que proporcionan funciones de manejo de errores configurando el Área de comunicaciones de SQL (SQLCA)

  Tenga en cuenta que las aplicaciones DB2 escritas en Java lanzan un `SQLException`, que debe manejar en un bloque `catch` en lugar de utilizando SQLCA.

Un programa puede contener varias secciones declare de SQL.

Procedimiento:

Para crear la sección de declaración:

1. Utilice la sentencia de SQL BEGIN DECLARE SECTION para abrir la sección.
2. Codifique las declaraciones.
3. Utilice la sentencia de SQL END DECLARE SECTION para finalizar la sección.

Tareas relacionadas:

- "Declaración de variables que interactúan con el gestor de bases de datos" en la página 29
- "Declaración de variables que representan objetos de SQL" en la página 30
- "Relación de variables del lenguaje principal con una sentencia de SQL" en la página 33
- "Declaración de variables del lenguaje principal con el Generador de declaraciones db2dclgn" en la página 32
- "Declaración de la SQLCA para el manejo de errores" en la página 34

Declaración de variables que interactúan con el gestor de bases de datos

Todas las variables que interactúan con el gestor de bases de datos se deben declarar en la sección de declaración de SQL.

Las variables de programa de lenguaje principal declaradas en una sección de declaración de SQL se denominan variables del lenguaje principal. Puede utilizar

variables del lenguaje principal en referencias a variables del lenguaje principal en sentencias de SQL. El código *variable-lenguaje-principal* se utiliza en diagramas de sintaxis en sentencias de SQL.

Procedimiento:

Para declarar una variable, codifíquela en la sección de declaración de SQL. A continuación se muestra un ejemplo de variable del lenguaje principal en C/C++:

```
EXEC SQL BEGIN DECLARE SECTION;
  short     dept=38, age=26;
  double    salary;
  char      CH;
  char      name1[9], NAME2[9];
  /* Comentario de C */
  short     nul_ind;
EXEC SQL END DECLARE SECTION;
```

Los atributos de cada variable del lenguaje principal dependen del modo en que se utiliza la variable en la sentencia de SQL. Por ejemplo, las variables que reciben datos de tablas de DB2 o que almacenan datos en las mismas deben tener atributos de tipo de datos y longitud compatibles con la columna a la que se accede. Para determinar el tipo de datos de cada variable, debe estar familiarizado con los tipos de datos de DB2.

Información relacionada:

- "Tipos de datos SQL soportados en C y C++" en la página 180
- "Tipos de datos de SQL soportados en COBOL" en la página 210
- "Tipos de datos SQL soportados en FORTRAN" en la página 227
- "Tipos de datos SQL soportados en REXX" en la página 539
- "Tipos de datos de Java, JDBC y SQL" en la página 395

Declaración de variables que representan objetos de SQL

Declare las variables que representan objetos de SQL en la sección de declaración de SQL del programa de aplicación.

Procedimiento:

Codifique la variable en el formato adecuado para el lenguaje en el que escribe el programa de aplicación.

Cuando codifique la variable, recuerde que los nombres de tablas, alias, vistas y correlaciones tienen una longitud máxima de 128 bytes. Los nombres de columna tienen una longitud máxima de 30 bytes. Los nombres de esquema tienen una longitud máxima de 30 bytes. Es posible que en futuros releases de DB2 aumenten las longitudes de nombres de columna y de otros identificadores de objetos SQL hasta 128 bytes. Si declara variables que representan objetos SQL con longitudes de menos de 128 bytes, futuros aumentos en longitudes de identificadores de objetos SQL pueden afectar a la estabilidad de las aplicaciones. Por ejemplo, si declara la variable `char[9]`*`schema_name`* en una aplicación C++ para que albergue un nombre de esquema, la aplicación funciona correctamente para los nombres de esquema permitidos en DB2 Versión 6, que tienen una longitud máxima de 8 bytes.

```
  char[9] schema_name; /* alberga nombre de esquema delimitado por nulos de hasta 8 bytes;
funciona para DB2 Versión 6, pero puede truncar nombres de esquema en futuros releases */
```

Sin embargo, si migra la base de datos a una versión de DB2 que acepta nombres de esquema con una longitud máxima de 30 bytes, la aplicación no puede diferenciar entre los nombres de esquema LONGSCHEMA1 y LONGSCHEMA2. El gestor de bases de datos trunca los nombres de esquema a su límite de 8 bytes, LONGSCHE, y cualquier sentencia de la aplicación que dependa de diferenciar los nombres de esquema fallará. Para aumentar la longevidad de la aplicación, declare la variable de nombre de esquema con una longitud de 128 bytes del siguiente modo:

```
char[129] schema_name; /* alberga nombre de esquema delimitado por nulos hasta 128 bytes
                          válido para DB2 Versión 7 y siguientes */
```

Para mejorar el futuro funcionamiento de la aplicación, considere la posibilidad de declarar todas las variables de la aplicación que representan nombres de objetos SQL con longitudes de 128 bytes. Debe sopesar la ventaja de mejorar la compatibilidad frente al aumento de recursos del sistema que necesitan las variables más largas.

Para aplicaciones C/C++, puede simplificar la codificación de declaraciones y aumentar la claridad del código si utiliza la expansión de macro C para declarar las longitudes de identificadores de objetos SQL. Puesto que el archivo include sql.h declara que SQL_MAX_IDENT sea 128, puede declarar fácilmente identificadores de objetos SQL con la macro SQL_MAX_IDENT. Por ejemplo:

```
#include <sql.h>
    char[SQL_MAX_IDENT+1] schema_name;
    char[SQL_MAX_IDENT+1] table_name;
    char[SQL_MAX_IDENT+1] employee_column;
    char[SQL_MAX_IDENT+1] manager_column;
```

Conceptos relacionados:

- “Variables del lenguaje principal en C y C++” en la página 152
- “Sintaxis de las variables del lenguaje principal de tipo carácter fijas y terminadas en nulo en C y C++” en la página 156
- “Expansión de macros en C” en la página 165
- “Variables del lenguaje principal en COBOL” en la página 198
- “Variables del lenguaje principal en FORTRAN” en la página 221
- “Variables del lenguaje principal en REXX” en la página 533

Información relacionada:

- “Sintaxis de las variables numéricas del lenguaje principal en C y C++” en la página 154
- “Sintaxis de las variables del lenguaje principal de tipo carácter de longitud variable en C o C++” en la página 157
- “Sintaxis de la declaración gráfica de formatos gráficos de un solo gráfico y terminados en nulo en C y C++” en la página 159
- “Sintaxis para la declaración gráfica del formato estructurado VARGRAPHIC en C o C++” en la página 160
- “Sintaxis de las variables del lenguaje principal de objeto grande (LOB) en C o C++” en la página 161
- “Sintaxis de las variables del lenguaje principal de localizador de objeto grande (LOB) en C o C++” en la página 163
- “Sintaxis de declaraciones de variables del lenguaje principal de referencia de archivos en C o C++” en la página 164
- “Sintaxis de las variables numéricas del lenguaje principal en COBOL” en la página 200

- "Sintaxis de las variables del lenguaje principal de tipo carácter de longitud fija en COBOL" en la página 201
- "Sintaxis de las variables gráficas del lenguaje principal de longitud fija en COBOL" en la página 203
- "Sintaxis de las variables del lenguaje principal LOB en COBOL" en la página 204
- "Sintaxis de las variables del lenguaje principal de localizador de LOB en COBOL" en la página 205
- "Sintaxis de las variables del lenguaje principal de referencia de archivos en COBOL" en la página 206
- "Sintaxis de las variables numéricas del lenguaje principal en FORTRAN" en la página 222
- "Sintaxis de las variables de tipo carácter del lenguaje principal en FORTRAN" en la página 223
- "Sintaxis de las variables del lenguaje principal de objeto grande (LOB) en FORTRAN" en la página 225
- "Sintaxis de las variables del lenguaje principal de localizador de objeto grande (LOB) en FORTRAN" en la página 226
- "Sintaxis de las variables del lenguaje principal de referencia de archivos en FORTRAN" en la página 226
- "Sintaxis de las declaraciones de localizador de LOB en REXX" en la página 537
- "Sintaxis de las declaraciones de referencia de archivos LOB en REXX" en la página 537

Declaración de variables del lenguaje principal con el Generador de declaraciones db2dclgn

Puede utilizar el Generador de declaraciones para generar declaraciones correspondientes a una determinada tabla de una base de datos. Éste crea archivos fuente de declaración en SQL incorporado que puede insertar fácilmente en las aplicaciones. `db2dclgn` da soporte a los lenguajes C/C++, Java, COBOL y FORTRAN.

Procedimiento:

Para generar archivos de declaración, entre el mandato `db2dclgn` en el siguiente formato:

```
db2dclgn -d database-name -t table-name [options]
```

Por ejemplo, para generar las declaraciones para la tabla STAFF en la base de datos SAMPLE en C en el archivo de salida `staff.h`, emita el siguiente mandato:

```
db2dclgn -d sample -t staff -l C
```

El archivo `staff.h` resultante contiene:

```
struct
{
  short id;
  struct
  {
    short length;
    char data[9];
  } name;
  short dept;
  char job[6];
```

```
        short years;
        double salary;
        double comm;
      } staff;
```

Información relacionada:

- "db2dclgn - Mandato Generador de declaraciones" en la publicación *Consulta de mandatos*

Relación de variables del lenguaje principal con una sentencia de SQL

Puede utilizar variables del lenguaje principal para recibir datos del gestor de bases de datos o para transferir datos al mismo desde el programa de lenguaje principal. Las variables del lenguaje principal que reciben datos del gestor de bases de datos son *variables del lenguaje principal de salida*, mientras que las que transfieren datos al mismo desde el programa de lenguaje principal son *variables del lenguaje principal de entrada*.

Considere la siguiente sentencia SELECT INTO:

```
SELECT HIREDATE, EDLEVEL
       INTO :hdate, :lvl
       FROM EMPLOYEE
       WHERE EMPNO = :idno
```

La sentencia contiene dos variables del lenguaje principal de salida, `hdate` y `lvl`, y una variable del lenguaje principal de entrada, `idno`. El gestor de bases de datos utiliza los datos almacenados en la variable del lenguaje principal `idno` para determinar el EMPNO de la fila que se recupera de la tabla EMPLOYEE. Si el gestor de bases de datos encuentra una fila que cumple con los criterios de búsqueda, `hdate` y `lvl` reciben los datos almacenados en las columnas HIREDATE y EDLEVEL, respectivamente. Esta sentencia ilustra una interacción entre el programa de lenguaje principal y el gestor de bases de datos utilizando columnas de la tabla EMPLOYEE.

Procedimiento:

Para definir la variable del lenguaje principal para utilizarla con una columna:

1. Averigüe el tipo de datos SQL correspondiente a dicha columna. Para ello, consulte el catálogo del sistema, que es un conjunto de vistas que contienen información sobre todas las tablas creadas en la base de datos.
2. Codifique las declaraciones adecuadas según el lenguaje principal.

   A cada columna de una tabla se asigna un tipo de datos en la definición CREATE TABLE. Debe relacionar este tipo de datos con el tipo de datos de lenguaje principal. Por ejemplo, el tipo de datos INTEGER es un entero con signo de 32 bits. Esto equivale a las siguientes entradas de descripción de datos en cada uno de los lenguajes principales, respectivamente:

   C/C++:
   ```
   sqlint32 variable_name;
   ```

   Java:
   ```
   int variable_name;
   ```

   COBOL:
   ```
   01  variable-name  PICTURE S9(9) COMPUTATIONAL-5.
   ```

FORTRAN:

```
INTEGER*4 variable_name
```

También puede utilizar el programa de utilidad de generación de declaraciones (`db2dclgn`) para generar las declaraciones adecuadas para una determinada tabla de una base de datos.

Conceptos relacionados:

- "Vistas de catálogo" en la publicación *Consulta de SQL, Volumen 1*

Tareas relacionadas:

- "Declaración de variables que interactúan con el gestor de bases de datos" en la página 29
- "Declaración de variables del lenguaje principal con el Generador de declaraciones db2dclgn" en la página 32
- "Creación de una sección de declaración de una aplicación autónoma" en la página 29

Información relacionada:

- "Tipos de datos SQL soportados en C y C++" en la página 180
- "Tipos de datos de SQL soportados en COBOL" en la página 210
- "Tipos de datos SQL soportados en FORTRAN" en la página 227
- "Tipos de datos SQL soportados en REXX" en la página 539
- "Tipos de datos de Java, JDBC y SQL" en la página 395

Declaración de la SQLCA para el manejo de errores

Puede declarar la SQLCA en el programa de aplicación para que el gestor de bases de datos pueda devolver información a la aplicación. Cuando preprocesa el programa, el gestor de bases de datos inserta las declaraciones de variables del lenguaje principal en lugar de la sentencia INCLUDE. El sistema se comunica con el programa utilizando las variables para distintivos de aviso, códigos de error e información de diagnóstico.

Después de ejecutar cada sentencia de SQL, el sistema devuelve un código de retorno en SQLCODE y en SQLSTATE. SQLCODE es un valor entero que resume la ejecución de la sentencia y SQLSTATE es un campo de carácter que proporciona códigos de error comunes entre los productos de bases de datos relacionales de IBM. SQLSTATE también cumple con los estándares ISO/ANS SQL92 y FIPS 127-2.

Nota: FIPS 127-2 hace referencia a *Federal Information Processing Standards Publication 127-2 for Database Language SQL*. ISO/ANS SQL92 hace referencia a *American National Standard Database Language SQL X3.135-1992* e *International Standard ISO/IEC 9075:1992, Database Language SQL*.

Observe que si SQLCODE es menor que 0, significa que se ha producido un error y que la sentencia no se ha procesado. Si SQLCODE es mayor que 0, significa que se ha emitido un aviso, pero la sentencia se procesa.

Para una aplicación DB2 escrita en C o C++, si la aplicación está formada por varios archivos fuente, sólo uno de los archivos debería incluir la sentencia EXEC SQL INCLUDE SQLCA para evitar varias definiciones de SQLCA. El resto de los archivos fuente deberían utilizar las siguientes líneas:

```
#include "sqlca.h"
extern struct sqlca sqlca;
```

Procedimiento:

Para declarar la SQLCA, codifique la sentencia `INCLUDE SQLCA` en el programa del siguiente modo:

- Para aplicaciones C o C++ utilice:
  ```
  EXEC SQL INCLUDE SQLCA;
  ```
- En aplicaciones Java no se utiliza de forma explícita la SQLCA. Utilice en su lugar los métodos de instancia `SQLException` para obtener los valores de SQLSTATE y SQLCODE.
- Para aplicaciones COBOL utilice:
  ```
  EXEC SQL INCLUDE SQLCA END-EXEC.
  ```
- Para aplicaciones FORTRAN utilice:
  ```
  EXEC SQL INCLUDE SQLCA
  ```

Si la aplicación debe cumplir con el estándar ISO/ANS SQL92 o FIPS 127-2, no utilice las sentencias anteriores ni la sentencia `INCLUDE SQLCA`.

Conceptos relacionados:

- "Manejo de errores utilizando la sentencia WHENEVER" en la página 35
- "Variables SQLSTATE y SQLCODE en C y C++" en la página 186
- "Variables SQLSTATE y SQLCODE en COBOL" en la página 213
- "Variables SQLSTATE y SQLCODE en FORTRAN" en la página 229
- "Variables SQLSTATE y SQLCODE en Perl" en la página 527

Tareas relacionadas:

- "Creación de una sección de declaración de una aplicación autónoma" en la página 29

Manejo de errores utilizando la sentencia WHENEVER

La sentencia WHENEVER hace que el precompilador genere código fuente que indica a la aplicación que vaya a una etiqueta especificada si se produce un error, un aviso o si no se encuentran filas durante la ejecución. La sentencia WHENEVER afecta a las siguientes sentencias de SQL ejecutables hasta que otra sentencia WHENEVER altera la situación.

La sentencia WHENEVER tiene tres formatos básicos:

```
EXEC SQL WHENEVER SQLERROR acción
EXEC SQL WHENEVER SQLWARNING acción
EXEC SQL WHENEVER NOT FOUND acción
```

En las sentencias anteriores:

SQLERROR
Identifica cualquier condición en la que SQLCODE < `0`.

SQLWARNING
Identifica cualquier condición en la que SQLWARN(0) = `W` o SQLCODE > `0` pero no es igual a `100`.

NOT FOUND
Identifica cualquier condición en la que SQLCODE = `100`.

En cada caso, *acción* puede ser cualquiera de las siguientes:

CONTINUE
Indica que hay que continuar con la siguiente instrucción de la aplicación.

GO TO *etiqueta*
Indica que hay que ir a la sentencia que va inmediatamente detrás de la etiqueta especificada después de GO TO. (GO TO puede especificarse como dos palabras o como una sola, GOTO.)

Si no se utiliza la sentencia WHENEVER, la acción por omisión consiste en continuar el proceso si se produce una condición de error, de aviso o de excepción durante la ejecución.

La sentencia WHENEVER debe aparecer antes de las sentencias de SQL que desea que se vean afectadas. De lo contrario, el precompilador no sabe que se debe generar código de manejo de errores adicional para las sentencias de SQL ejecutables. En un momento cualquiera puede estar activa una combinación cualquiera de los tres formatos básicos. El orden en el que declare los tres formatos no es significativo.

Para evitar una situación de bucle infinito, asegúrese de deshacer el manejo WHENEVER antes de que se ejecute alguna sentencia de SQL dentro del manejador. Puede hacerlo utilizando la sentencia WHENEVER SQLERROR CONTINUE.

Información relacionada:

- "Sentencia WHENEVER" en la publicación *Consulta de SQL, Volumen 2*

Adición de sentencias no ejecutables a una aplicación

Si tiene que incluir sentencias de SQL no ejecutables en un programa de aplicación, normalmente puede colocarlas en la sección de declaración de la aplicación. Las sentencias INCLUDE, INCLUDE SQLDA y DECLARE CURSOR son ejemplos de sentencias no ejecutables.

Procedimiento:

Si desea utilizar la sentencia no ejecutable INCLUDE en la aplicación, codifíquela del siguiente modo:

```
INCLUDE nombre-archivo-texto
```

Tareas relacionadas:

- "Creación de una sección de declaración de una aplicación autónoma" en la página 29

Conexión de una aplicación con una base de datos

El programa debe establecer una conexión con la base de datos de destino para que pueda ejecutar las sentencias de SQL ejecutables. Esta conexión identifica el ID de autorización del usuario que está ejecutando el programa y el nombre del servidor de bases de datos en el que se ejecuta el programa. Generalmente, el proceso de la aplicación sólo puede conectarse a un servidor de bases de datos cada vez. Este servidor se denomina el *servidor actual*. Sin embargo, la aplicación puede conectarse a varios servidores de bases de datos dentro de un entorno de actualización múltiple. En este caso, sólo un servidor puede ser el servidor actual.

Restricciones:

Se aplican las siguientes restricciones:

- Una conexión dura hasta que se emite una sentencia CONNECT RESET, CONNECT TO o DISCONNECT.
- En un entorno de actualización múltiple, una conexión también dura hasta que se emite DB2 RELEASE y luego DB2 COMMIT. Una sentencia CONNECT TO no termina una conexión cuando se utiliza la actualización múltiple.

Procedimiento:

El programa puede establecer una conexión con un servidor de bases de datos:

- De forma explícita, mediante la sentencia CONNECT
- De forma implícita, estableciendo conexión con el servidor de bases de datos por omisión
- Para aplicaciones Java, mediante una instancia `Connection`

Consulte la descripción de la sentencia CONNECT para obtener información sobre estados de conexión y para ver cómo se utiliza la sentencia CONNECT. Tras la inicialización, el solicitante de la aplicación establece un servidor de bases de datos por omisión. Si hay conexiones implícitas habilitadas, los procesos de la aplicación iniciados tras la inicialización se conectan de forma implícita al servidor de bases de datos por omisión. Es recomendable utilizar la sentencia CONNECT como primera sentencia de SQL ejecutada por un programa de aplicación. Una sentencia CONNECT explícita evita la ejecución accidental de sentencias de SQL contra la base de datos por omisión.

Conceptos relacionados:

- "Actualización múltiple" en la página 672

Información relacionada:

- "Sentencia CONNECT (Tipo 1)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CONNECT (Tipo 2)" en la publicación *Consulta de SQL, Volumen 2*

Codificación de transacciones

Una transacción es una secuencia de sentencias de SQL (posiblemente con código de lenguaje de sistema principal) que el gestor de bases de datos trata como una unidad. Un término alternativo que se utiliza a menudo para transacción es *unidad de trabajo*.

Requisitos previos:

Se debe establecer una conexión con la base de datos contra la que se va a ejecutar la transacción.

Procedimiento:

Para codificar una transacción:

1. Inicie la transacción con una sentencia de SQL *ejecutable*.

   Una vez establecida la conexión con la base de datos, el programa puede emitir una o más:

   - Sentencias de manipulación de datos (por ejemplo, la sentencia SELECT)

- Sentencias de definición de datos (por ejemplo, la sentencia CREATE)
- Sentencias de control de datos (por ejemplo, la sentencia GRANT)

Una sentencia de SQL ejecutable siempre se produce dentro de una transacción. Si un programa contiene una sentencia de SQL ejecutable después de que finalice una transacción, inicia automáticamente una nueva transacción.

Nota: Las seis sentencias siguientes *no* inician una transacción porque no son sentencias ejecutables:
- BEGIN DECLARE SECTION
- INCLUDE SQLCA
- END DECLARE SECTION
- INCLUDE SQLDA
- DECLARE CURSOR
- WHENEVER

2. Finalice la transacción de una de las siguientes formas:
   - Confirme (COMMIT) la transacción
   - Retrotraiga (ROLLBACK) la transacción

Tareas relacionadas:

- "Finalización de una transacción con la sentencia COMMIT" en la página 38
- "Finalización de una transacción con la sentencia ROLLBACK" en la página 39

Finalización de una transacción con la sentencia COMMIT

La sentencia COMMIT finaliza la transacción actual y hace que los cambios en la base de datos realizados durante la transacción estén visibles para otros procesos.

Procedimiento:

Confirme los cambios en cuanto los requisitos de la aplicación lo permitan. En concreto, escriba los programas de modo que los cambios no confirmados no se mantengan mientras se espere una entrada de un terminal, puesto que esto podría ocasionar que los recursos de la base de datos se mantuvieran durante mucho tiempo. Al mantener estos recursos se evita la ejecución de otras aplicaciones que necesitan dichos recursos.

Los programas de aplicación deben finalizar de forma explícita cualquier transacción antes de terminar.

Si no finaliza las transacciones de forma explícita, DB2 confirma automáticamente todos los cambios realizados durante la transacción pendiente del programa cuando el programa finaliza satisfactoriamente, excepto en sistemas operativos Windows. En sistemas operativos Windows, si no confirma de forma explícita una transacción, el gestor de bases de datos siempre retrotrae los cambios.

DB2 retrotrae los cambios bajo las siguientes condiciones:
- Una condición de registro lleno
- Cualquier otra condición del sistema que haga que finalice el proceso del gestor de bases de datos

La sentencia COMMIT no tiene ningún efecto sobre el contenido de las variables del lenguaje principal.

Conceptos relacionados:

- "Finalización implícita de una transacción en una aplicación autónoma" en la página 40
- "Códigos de retorno" en la página 110
- "Información de error en los campos SQLCODE, SQLSTATE y SQLWARN" en la página 110

Tareas relacionadas:

- "Finalización de un programa de aplicación" en la página 40

Información relacionada:

- "Sentencia COMMIT" en la publicación *Consulta de SQL, Volumen 2*

Finalización de una transacción con la sentencia ROLLBACK

Para garantizar la coherencia de datos en una transacción, el sistema gestor de bases de datos se asegura de que *todas* las operaciones de una transacción se completan o de que *ninguna* se completa. Supongamos, por ejemplo, que el programa tiene que deducir dinero de una cuenta y añadirlo a otra. Si coloca estas dos actualizaciones en una sola transacción, y se produce un error del sistema mientras se están procesando, cuando vuelva a iniciar el sistema el gestor de bases de datos realiza automáticamente una recuperación de error para restaurar los datos al estado en que estaban antes de que comenzara la transacción. Si se produce un error del programa, el gestor de bases de datos restaura todos los cambios realizados por la sentencia que ha ocasionado el error. El gestor de bases de datos no deshará el trabajo realizado en la transacción antes de la ejecución de la sentencia errónea, a no ser que la retrotraiga de forma específica.

Procedimiento:

Para evitar que los cambios afectados por la transacción se confirmen en la base de datos, emita la sentencia ROLLBACK para finalizar la transacción. La sentencia ROLLBACK devuelve la base de datos al estado en que estaba antes de que se ejecutara la transacción.

Nota: En sistemas operativos Windows, si no confirma de forma explícita una transacción, el gestor de bases de datos siempre retrotrae los cambios.

Si utiliza la sentencia ROLLBACK en una rutina en la que se entró debido a un error o aviso y utiliza la sentencia de SQL WHENEVER, debe especificar WHENEVER SQLERROR CONTINUE y WHENEVER SQLWARNING CONTINUE antes de ROLLBACK. Esto evita un bucle del programa si la sentencia ROLLBACK falla con un error o aviso.

En el caso de un error grave, recibirá un mensaje que indicará que no puede emitir una sentencia ROLLBACK. No emita una sentencia ROLLBACK si se produce un error grave, como una pérdida de comunicaciones entre las aplicaciones cliente y servidor o si la base de datos resulta dañada. Después de un error grave, la única sentencia que puede emitir es una sentencia CONNECT.

La sentencia ROLLBACK no tiene ningún efecto sobre el contenido de las variables del lenguaje principal.

Puede codificar una o más transacciones dentro de un solo programa de aplicación, y se puede acceder a más de una base de datos desde una sola transacción. Una transacción que accede a más de una base de datos se denomina actualización múltiple.

Conceptos relacionados:

- "Finalización implícita de una transacción en una aplicación autónoma" en la página 40
- "Unidad de trabajo remota" en la página 671
- "Actualización múltiple" en la página 672

Información relacionada:

- "Sentencia CONNECT (Tipo 1)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CONNECT (Tipo 2)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia WHENEVER" en la publicación *Consulta de SQL, Volumen 2*

Finalización de un programa de aplicación

Finalice un programa de aplicación para liberar los recursos que utilizaba el programa.

Procedimiento:

Para finalizar correctamente el programa:

1. Finalice la transacción actual (si se está procesando alguna) emitiendo de forma explícita una sentencia COMMIT o ROLLBACK.
2. Libere la conexión con el servidor de bases de datos utilizando la sentencia CONNECT RESET.
3. Libere los recursos utilizados por el programa. Por ejemplo, libere el almacenamiento temporal o las estructuras de datos utilizadas.

Nota: Si la transacción actual sigue activa cuando termina el programa, DB2 finaliza de forma explícita la transacción. Puesto que el comportamiento de DB2 cuando finaliza de forma implícita una transacción depende de cada plataforma, debe finalizar de forma explícita todas las transacciones emitiendo una sentencia COMMIT o ROLLBACK antes de que termine el programa.

Conceptos relacionados:

- "Finalización implícita de una transacción en una aplicación autónoma" en la página 40

Información relacionada:

- "Sentencia CONNECT (Tipo 1)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CONNECT (Tipo 2)" en la publicación *Consulta de SQL, Volumen 2*

Finalización implícita de una transacción en una aplicación autónoma

Si el programa termina sin finalizar la transacción actual, DB2® termina de forma implícita la transacción actual. DB2 termina de forma implícita la transacción

actual emitiendo una sentencia COMMIT o ROLLBACK y luego la aplicación finaliza. Si la emisión de una sentencia COMMIT o ROLLBACK por parte de DB2 depende de factores como los siguientes:

- Si la aplicación ha terminado normalmente

  En la mayoría de los sistemas operativos soportados, DB2 confirma de forma implícita una transacción si la terminación es normal o retrotrae de forma implícita la transacción si la terminación es anómala. Tenga en cuenta que lo que el programa considera una terminación anómala puede no ser considerado como tal por el gestor de bases de datos. Por ejemplo, puede codificar `exit(-16)` cuando la aplicación encuentre un error inesperado y terminar la aplicación de inmediato. El gestor de bases de datos considera que esto es una terminación normal y confirma la transacción. El gestor de bases de datos considera elementos tales como una excepción o una infracción de segmentación como terminaciones anómalas.
- La plataforma en la que se ejecuta el servidor DB2

  En sistemas operativos Windows® de 32 bits, DB2 siempre retrotrae la transacción, independientemente de si la aplicación termina de forma normal o anómala. Si desea que la transacción se confirme, debe emitir la sentencia COMMIT de forma explícita.
- Si la aplicación utiliza las API de contexto de DB2 para el acceso a bases de datos de varias hebras

  Si la aplicación las utiliza, DB2 retrotrae de forma implícita la transacción tanto si la aplicación termina de forma normal como si lo hace de forma anómala. A no ser que confirme de forma explícita la transacción utilizando la sentencia COMMIT, DB2 retrotrae la transacción.

Conceptos relacionados:

- "Objetivo del acceso a base de datos de varias hebras" en la página 187

Tareas relacionadas:

- "Finalización de un programa de aplicación" en la página 40

Información relacionada:

- "Sentencia COMMIT" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia ROLLBACK" en la publicación *Consulta de SQL, Volumen 2*

Infraestructura de seudocódigo de aplicación

El siguiente ejemplo resume la infraestructura general correspondiente a un programa de aplicación de DB2 en formato de seudocódigo. Por supuesto, debe adaptar esta infraestructura para que se ajuste a su propio programa.

```
Iniciar programa
EXEC SQL BEGIN DECLARE SECTION                                  |
  DECLARE USERID FIXED CHARACTER (8)                            |
  DECLARE PW FIXED CHARACTER (8)                                |
                                                                | Configuración
  (otras declaraciones de variables del lenguaje principal)     | de aplicación
                                                                |
EXEC SQL END DECLARE SECTION                                    |
EXEC SQL INCLUDE SQLCA                                          |
EXEC SQL WHENEVER SQLERROR GOTO ERRCHK                          |

  (lógica de programa)

EXEC SQL CONNECT TO base_de_datos A USER :id_usuario USING :pw|
```

```
EXEC SQL SELECT ...                                                  |
EXEC SQL INSERT ...                                                  | Primera unidad
  (más sentencias de SQL)                                            | de trabajo
EXEC SQL COMMIT                                                      |

  (más lógica de programa)

EXEC SQL CONNECT TO base_de_datos B USER :id_usuario USING :pw       |
EXEC SQL SELECT ...                                                  |
EXEC SQL DELETE ...                                                  | Segunda unidad
  (más sentencias de SQL)                                            | de trabajo
EXEC SQL COMMIT                                                      |

  (más lógica de programa)

EXEC SQL CONNECT TO base_de_datos A                                  |
EXEC SQL SELECT ...                                                  |
EXEC SQL DELETE ...                                                  | Tercera unidad
  (más sentencias de SQL)                                            | de trabajo
EXEC SQL COMMIT                                                      |

  (más lógica de programa)

EXEC SQL CONNECT RESET                                               |
ERRCHK                                                               |
                                                                     | Limpieza de
  (comprobar información de error en SQLCA)                          | aplicación
                                                                     |
Finalizar programa
```

Tareas relacionadas:

• "Programación de una aplicación autónoma" en la página 28

Recursos para crear prototipos de sentencias de SQL

A medida que diseña y codifica la aplicación, puede aprovechar determinadas funciones y programas de utilidad del gestor de bases de datos para crear prototipos de partes del código SQL y para mejorar el rendimiento. Por ejemplo, puede hacer lo siguiente:

- Utilizar el Centro de control o el procesador de línea de mandatos (CLP) para probar muchas sentencias de SQL antes de intentar compilar y enlazar un programa completo.

  Esto le permite definir y manipular información almacenada en una tabla de base de datos, índice o vista. Puede añadir, suprimir o actualizar información, así como generar informes a partir del contenido de las tablas. Observe que tiene que modificar mínimamente la sintaxis de algunas sentencias de SQL para utilizar variables del lenguaje principal en el programa de SQL incorporado. Las variables del lenguaje principal se utilizan para almacenar datos que representan salida en la pantalla. Además, algunas sentencias de SQL incorporado (como BEGIN DECLARE SECTION) no reciben soporte del Centro de mandatos ni de CLP puesto que no resultan relevantes para dicho entorno.

  También puede redirigir la entrada y la salida de las peticiones del procesador de línea de mandatos. Por ejemplo, podría crear uno o más archivos que contengan sentencias de SQL que necesita como entrada en una petición del procesador de línea de mandatos para no tener que volver a escribir la sentencia.
- Utilizar el recurso Explain para obtener una idea de los costes estimados de las sentencias DELETE, INSERT, UPDATE o SELECT que tiene intención de utilizar en el programa. El recurso Explain coloca la información sobre la estructura y

los costes estimados de la sentencia en cuestión en tablas suministradas por el usuario. Puede ver esta información utilizando Visual Explain o el programa de utilidad `db2exfmt`.

- Utilizar las vistas del catálogo del sistema para recuperar fácilmente información sobre las bases de datos existentes. El gestor de bases de datos crea y mantiene las tablas del catálogo del sistema en las que se basan las vistas durante el funcionamiento normal a medida que las bases de datos se crean, modifican y actualizan. Estas vistas contienen datos sobre cada base de datos, que incluyen autorizaciones otorgadas, nombres de columna, tipos de datos, índices, dependencias de paquetes, restricciones referenciales, nombres de tabla, vistas, etc. Los datos de las vistas del catálogo del sistema están disponibles a través de los recursos de consulta normales de SQL.

  Puede actualizar algunas vistas del catálogo del sistema que contienen información estadística que utiliza el optimizador de SQL. Puede cambiar algunas columnas de estas vistas para que afecten al optimizador o para investigar el rendimiento de bases de datos hipotéticas. Puede utilizar este método para simular un sistema de producción en su sistema de desarrollo o prueba y analizar el comportamiento de las consultas.

Conceptos relacionados:

- "Vistas de catálogo" en la publicación *Consulta de SQL, Volumen 1*
- "Catalog statistics tables" en la publicación *Administration Guide: Performance*
- "Catalog statistics for modeling and what-if planning" en la publicación *Administration Guide: Performance*
- "General rules for updating catalog statistics manually" en la publicación *Administration Guide: Performance*
- "SQL explain facility" en la publicación *Administration Guide: Performance*
- "Herramientas de DB2 Universal Database para desarrollar aplicaciones" en la página 3

Información relacionada:

- Apéndice A, "Sentencias de SQL soportadas", en la página 733

API administrativas en SQL incorporado o en programas de CLI de DB2

La aplicación puede utilizar API para acceder a las funciones del gestor de bases de datos que no están disponibles utilizando sentencias de SQL.

Puede utilizar las API de DB2® para:

- Manipular el entorno del gestor de bases de datos, que incluye la catalogación y descatalogación de bases de datos y nodos y la exploración de directorios de bases de datos y de nodos. También puede utilizar API para crear, suprimir y migrar bases de datos.
- Proporcionar recursos para importar y exportar datos y administrar, hacer copia de seguridad y restaurar la base de datos.
- Modificar los valores de parámetros de configuración del gestor de bases de datos y de la base de datos.
- Proporcionar operaciones específicas del entorno cliente/servidor.

- Proporcionar la interfaz en tiempo de ejecución para sentencias de SQL precompiladas. El desarrollador no suele llamar directamente a estas API. En su lugar, el precompilador las inserta en el archivo fuente modificado después del proceso.

El gestor de bases de datos incluye API para proveedores de lenguajes que desean escribir su propio precompilador y otras API útiles para desarrollar aplicaciones.

Conceptos relacionados:

- "Consideraciones sobre autorización para API" en la página 53

Control de valores de datos y relaciones

Las secciones siguientes describen cómo controlar valores de datos y relaciones entre los datos.

Control de valores de datos

Un área tradicional de la lógica de aplicación es validar y proteger la integridad de los datos, controlando los valores permitidos en la base de datos. Las aplicaciones tienen lógica que comprueba de forma específica la validez de los valores de datos a medida que se entran. Por ejemplo, comprobación de que el número de departamento es un número válido y de que hace referencia a un departamento existente. Existen varias formas de proporcionar estas mismas funciones en DB2®, pero desde dentro de la base de datos.

Conceptos relacionados:

- "Control de valores de datos mediante tipos de datos" en la página 44
- "Control de valores de datos mediante restricciones exclusivas" en la página 45
- "Control de valores de datos mediante restricciones de comprobación de tabla" en la página 45
- "Control de valores de datos mediante restricciones de integridad referencial" en la página 45
- "Control de valores de datos mediante vistas con la opción Check" en la página 46
- "Control de valores de datos mediante lógica de aplicación y tipos de variables de programa" en la página 46

Control de valores de datos mediante tipos de datos

La base de datos almacena cada elemento de datos en una columna de una tabla y define cada columna con un tipo de datos. Este tipo de datos establece ciertos límites en los tipos de valores para la columna. Por ejemplo, un entero debe ser un número comprendido dentro de un rango fijo. El uso de la columna en sentencias de SQL debe cumplir con determinados comportamientos; por ejemplo, la base de datos no compara un entero con una serie de caracteres. DB2® incluye un grupo de tipos de datos integrados con características y comportamientos definidos. DB2 también da soporte a la definición de sus propios datos, denominados *tipos diferenciados definidos por el usuario*, que se basan en los tipos integrados pero no dan soporte automáticamente a todos los comportamientos del tipo integrado. También puede utilizar tipos de datos, como objeto grande binario (BLOB), para almacenar datos que pueden consistir en un grupo de valores relacionados, como una estructura de datos.

Conceptos relacionados:

- "Tipos diferenciados definidos por el usuario" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*

Control de valores de datos mediante restricciones exclusivas

Las restricciones exclusivas evitan la presencia de valores duplicados en una o más columnas dentro de una tabla. Las claves exclusivas y primarias son las restricciones exclusivas soportadas. Por ejemplo, puede definir una restricción exclusiva en la columna DEPTNO de la tabla DEPARTMENT para asegurar que no se asigna el mismo número de departamento a dos departamentos.

Utilice restricciones exclusivas si necesita aplicar la norma de exclusividad para todas las aplicaciones que utilizan los datos de una tabla.

Tareas relacionadas:

- "Defining a unique constraint" en la publicación *Administration Guide: Implementation*
- "Adding a unique constraint" en la publicación *Administration Guide: Implementation*

Control de valores de datos mediante restricciones de comprobación de tabla

Puede utilizar una restricción de comprobación de tabla para definir restricciones, además de las del tipo de datos, sobre los valores permitidos para una columna de una tabla. Las restricciones de comprobación de tabla adoptan la forma de comprobaciones de rango o comparaciones con otros valores de la misma fila de la misma tabla.

Si la norma se aplica a todas las aplicaciones que utilizan los datos, utilice una restricción de comprobación de tabla para aplicar la restricción sobre los datos permitidos en la tabla. Las restricciones de comprobación de tabla hacen que la restricción se aplique de forma general y sea más fácil de mantener.

Tareas relacionadas:

- "Defining a table check constraint" en la publicación *Administration Guide: Implementation*
- "Adding a table check constraint" en la publicación *Administration Guide: Implementation*

Control de valores de datos mediante restricciones de integridad referencial

Utilice las restricciones de integridad referencial (RI) si debe mantener relaciones basadas en valores para todas las aplicaciones que utilizan los datos. Por ejemplo, puede utilizar una restricción RI para asegurar que el valor de una columna DEPTNO de una tabla EMPLOYEE coincide con un valor de la tabla DEPARTMENT. Esta restricción evita inserciones, actualizaciones o supresiones que darían lugar a la pérdida de información de DEPARTMENT. Al centralizar las normas en la base de datos, las restricciones RI hacen que las normas se apliquen de forma general y sean más fáciles de mantener.

Conceptos relacionados:

- "Restricciones" en la publicación *Consulta de SQL, Volumen 1*
- "Control de relaciones de datos mediante restricciones de integridad referencial" en la página 47
- "Diferencias en la integridad referencial entre sistemas de bases de datos relacionales de IBM" en la página 745

Control de valores de datos mediante vistas con la opción Check

Si la aplicación no puede definir las normas deseadas como restricciones de comprobación de tabla, o si las normas no se aplican a todos los usos de los datos, hay otra alternativa a colocar las normas en la lógica de la aplicación. Puede considerar la posibilidad de crear una vista de la tabla con las condiciones sobre los datos como parte de la cláusula WHERE y la cláusula WITH CHECK OPTION especificada. Esta definición de vista restringe la recuperación de datos al conjunto que es válido para su aplicación. Además, si puede actualizar la vista, la cláusula WITH CHECK OPTION restringe actualizaciones, inserciones y supresiones en las filas que se aplican a su aplicación.

Información relacionada:

- "Sentencia CREATE VIEW" en la publicación *Consulta de SQL, Volumen 2*

Control de valores de datos mediante lógica de aplicación y tipos de variables de programa

Cuando escribe su lógica de aplicación en un lenguaje de programación, también declara variables para proporcionar algunas de las mismas restricciones sobre los datos que se han descrito en otros temas sobre control de valores de datos. Además, puede escribir código que haga cumplir normas en la aplicación en lugar de en la base de datos. Coloque la lógica en el servidor de aplicaciones cuando:

- Las normas no se apliquen de forma general, excepto en el caso de vistas que utilizan la opción WITH CHECK OPTION
- No tenga control sobre las definiciones de los datos en la base de datos
- Crea que la norma puede ser más eficiente si se maneja en la lógica de aplicación

Por ejemplo, es posible que se tengan que procesar errores en los datos de entrada en el orden en que se entran, pero no se puede garantizar el orden de las operaciones dentro de la base de datos.

Conceptos relacionados:

- "Control de valores de datos mediante vistas con la opción Check" en la página 46

Control de relaciones de datos

Una de las principales áreas de interés de la lógica de aplicación es la gestión de las relaciones entre las distintas entidades lógicas del sistema. Por ejemplo, si añade un nuevo departamento, tiene que crear un nuevo código de cuenta. DB2® proporciona dos métodos para gestionar las relaciones entre distintos objetos en la base de datos: restricciones de integridad referencial y activadores.

Conceptos relacionados:

- "Control de relaciones de datos mediante restricciones de integridad referencial" en la página 47
- "Control de relaciones de datos mediante activadores" en la página 47
- "Control de relaciones de datos mediante activadores anteriores" en la página 48
- "Control de relaciones de datos mediante activadores posteriores" en la página 48
- "Control de relaciones de datos mediante lógica de aplicación" en la página 49

Control de relaciones de datos mediante restricciones de integridad referencial

Las restricciones de integridad referencial (RI), consideradas desde la perspectiva del control de relaciones de datos, le permiten controlar las relaciones entre datos en más de una tabla. Utilice las sentencias CREATE TABLE o ALTER TABLE para definir el comportamiento de operaciones que afectan a la clave primaria relacionada, como DELETE y UPDATE.

Las restricciones RI hacen cumplir las normas sobre los datos en una o más tablas. Si las normas se aplican para todas las aplicaciones que utilizan los datos, las restricciones RI centralizan las normas en la base de datos. Esto hace que las normas se apliquen de forma general y sean más fáciles de mantener.

Conceptos relacionados:

- "Restricciones" en la publicación *Consulta de SQL, Volumen 1*

Tareas relacionadas:

- "Defining referential constraints" en la publicación *Administration Guide: Implementation*

Información relacionada:

- "Sentencia ALTER TABLE" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE TABLE" en la publicación *Consulta de SQL, Volumen 2*

Control de relaciones de datos mediante activadores

Puede utilizar activadores antes o después de una actualización para dar soporte a la lógica que también se puede llevar a cabo en una aplicación. Si las normas u operaciones que soportan los activadores se aplican a todas las aplicaciones que utilizan los datos, los activadores centralizan las normas u operaciones en la base de datos, lo que hace que estén disponibles de forma general y sean más fáciles de mantener.

Conceptos relacionados:

- "Control de relaciones de datos mediante activadores anteriores" en la página 48
- "Control de relaciones de datos mediante activadores posteriores" en la página 48
- "Activadores de DB2" en la página 24

Tareas relacionadas:

- "Creating a trigger" en la publicación *Administration Guide: Implementation*

- "Creación de activadores" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*

Información relacionada:

- "Sentencia CREATE TRIGGER" en la publicación *Consulta de SQL, Volumen 2*

Control de relaciones de datos mediante activadores anteriores

Si se utilizan activadores que se ejecutan antes de una actualización o inserción, los valores que se están actualizando o insertando se pueden modificar antes de que la base de datos se modifique realmente. Estos se pueden utilizar para transformar entrada procedente de la aplicación (vista de usuario de los datos) en un formato de base de datos interno, si se desea. Estos *activadores anteriores* también se pueden utilizar para hacer que se activen otras operaciones que no son de base de datos a través de funciones definidas por el usuario.

Conceptos relacionados:

- "Control de relaciones de datos mediante activadores posteriores" en la página 48
- "Activadores de DB2" en la página 24

Tareas relacionadas:

- "Creating a trigger" en la publicación *Administration Guide: Implementation*
- "Creación de activadores" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*

Información relacionada:

- "Sentencia CREATE TRIGGER" en la publicación *Consulta de SQL, Volumen 2*

Control de relaciones de datos mediante activadores posteriores

Los activadores que se ejecutan después de una actualización, inserción o supresión se pueden utilizar de varias formas:

- Los activadores pueden actualizar, insertar o suprimir datos en la misma tabla o en otras tablas. Esto resulta útil para mantener relaciones entre datos o para mantener información de seguimiento de auditoría.
- Los activadores pueden comparar datos con valores de datos en el resto de la tabla o en otras tablas. Esto resulta útil cuando no puede utilizar restricciones RI ni restricciones de comprobación debido a las referencias a datos desde otras filas de esta o de otras tablas.
- Los activadores pueden utilizar funciones definidas por el usuario para activar operaciones que no son de base de datos. Esto resulta útil, por ejemplo, para emitir alertas o actualizar información fuera de la base de datos.

Conceptos relacionados:

- "Control de relaciones de datos mediante activadores anteriores" en la página 48
- "Activadores de DB2" en la página 24

Tareas relacionadas:

- "Creating a trigger" en la publicación *Administration Guide: Implementation*

- "Creación de activadores" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*

Información relacionada:

- "Sentencia CREATE TRIGGER" en la publicación *Consulta de SQL, Volumen 2*

Control de relaciones de datos mediante lógica de aplicación

Es posible que decida escribir código para aplicar normas o realizar operaciones relacionadas en la aplicación en lugar de en la base de datos. Debe hacerlo en los casos en los que no puede aplicar de forma general las normas a la base de datos. También puede elegir el hecho de colocar la lógica en la aplicación si no tiene control sobre las definiciones de los datos de la base de datos o si cree que la lógica de aplicación puede manejar las normas u operaciones de forma más eficiente.

Conceptos relacionados:

- "Lógica de aplicación en el servidor" en la página 49

Lógica de aplicación en el servidor

Un último aspecto del diseño de aplicaciones para el que DB2® ofrece funciones adicionales es la ejecución de parte de la lógica de aplicación en el servidor de bases de datos. Generalmente, elegirá este diseño para mejorar el rendimiento, pero también puede ejecutar lógica de aplicación en el servidor para dar soporte a funciones comunes.

Puede utilizar lo siguiente:

- Procedimientos almacenados

  Un procedimiento almacenado es una rutina para la aplicación a la que se llama desde la lógica de aplicación cliente, pero se ejecuta en el servidor de bases de datos. La razón más común para utilizar un procedimiento almacenado es para el proceso intensivo de bases de datos que sólo genera una pequeña cantidad de datos de resultados. Esto puede ahorrar una gran cantidad de comunicaciones a través de la red durante la ejecución del procedimiento almacenado. También puede considerar la posibilidad de utilizar un procedimiento almacenado para un conjunto de operaciones que son comunes para varias aplicaciones. De este modo, todas las aplicaciones utilizan la misma lógica para llevar a cabo la operación.
- Funciones definidas por el usuario

  Puede escribir una función definida por el usuario (UDF) para utilizarla para llevar a cabo operaciones dentro de una sentencia de SQL para que devuelva:

  – Un solo valor escalar (función escalar)
  – Una tabla procedente de una fuente de datos que no sea DB2, por ejemplo un archivo ASCII o una página Web (función de tabla)

  Las UDF resultan útiles para tareas como transformar valores de datos, realizar cálculos sobre uno o más valores de datos o extraer partes de un valor (como extraer partes de un objeto grande).
- Activadores

  Se pueden utilizar activadores para invocar funciones definidas por el usuario. Esto resulta útil si desea que siempre se lleve a cabo una determinada operación no SQL cuando se produzcan determinadas sentencias o se cambien valores de

datos. Ejemplos tales son operaciones como emitir un mensaje de correo electrónico bajo circunstancias específicas o grabar información de tipo de alerta en un archivo.

Conceptos relacionados:

- "Control de relaciones de datos mediante activadores anteriores" en la página 48
- "Control de relaciones de datos mediante activadores posteriores" en la página 48
- "Guidelines for stored procedures" en la publicación *Administration Guide: Performance*
- "Interacciones de los activadores con las restricciones referenciales" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*
- "Procedimientos almacenados de DB2" en la página 19
- "Métodos y funciones definidas por el usuario de DB2" en la página 20
- "Activadores de DB2" en la página 24

Tareas relacionadas:

- "Creating a trigger" en la publicación *Administration Guide: Implementation*
- "Creación de activadores" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*

Información relacionada:

- "Sentencia CREATE TRIGGER" en la publicación *Consulta de SQL, Volumen 2*

Consideraciones sobre autorización para SQL y API

Las secciones siguientes describen las consideraciones generales sobre autorización para SQL incorporado y las consideraciones sobre autorización para SQL estático y dinámico y para las API.

Consideraciones sobre autorización para SQL incorporado

Una *autorización* permite a un usuario o grupo llevar a cabo una tarea general como conectarse a una base de datos, crear tablas o administrar un sistema. Un *privilegio* otorga a un usuario o grupo el derecho a acceder a un objeto de base de datos específico de una forma especificada. DB2® utiliza un grupo de privilegios para proteger la información que almacena en el mismo.

La mayoría de las sentencias de SQL necesitan algún tipo de privilegio sobre los objetos de base de datos que utiliza la sentencia. La mayoría de llamadas a API no suelen necesitar ningún privilegio sobre los objetos de base de datos que utiliza la llamada, aunque muchas API necesitan que el usuario tenga la autorización necesaria para invocarlas. Las API de DB2 le permiten realizar funciones administrativas de DB2 desde dentro del programa de aplicación. Por ejemplo, para recrear un paquete almacenado en la base de datos sin necesidad de disponer de un archivo de vinculación, puede utilizar la API `sqlarbnd` (o REBIND).

Cuando diseñe la aplicación, tenga en cuenta los privilegios que necesitarán los usuarios para ejecutar la aplicación. Los privilegios que necesitan los usuarios dependen de:

- Si la aplicación utiliza SQL dinámico, incluidos JDBC y CLI de DB2, o SQL estático. Para obtener información sobre los privilegios necesarios para emitir una sentencia, consulte la descripción de dicha sentencia.
- Qué API utiliza la aplicación. Para obtener información sobre las autorizaciones y los privilegios necesarios para una llamada a API, consulte la descripción de dicha API.

Supongamos que tenemos dos usuarios, PAYROLL y BUDGET, que tienen que realizar consultas contra la tabla STAFF. PAYROLL es responsable de pagar a los empleados de la empresa, de modo que tiene que emitir varias sentencias SELECT al emitir cheques. PAYROLL tiene que poder acceder al salario de cada empleado. BUDGET es responsable de determinar la cantidad de dinero necesaria para pagar los salarios. Sin embargo, BUDGET no debería poder ver el salario de ningún empleado en particular.

Puesto que PAYROLL emite muchas sentencias SELECT diferentes, la aplicación que diseñe para PAYROLL probablemente podría utilizar de forma eficiente código SQL dinámico. El código SQL dinámico necesitaría que PAYROLL tuviera el privilegio SELECT sobre la tabla STAFF. Este requisito no representa un problema porque PAYROLL necesita acceso completo a la tabla.

Por otro lado, BUDGET no debería tener acceso al salario de cada empleado. Esto significa que no debe otorgar el privilegio SELECT sobre la tabla STAFF a BUDGET. Puesto que BUDGET necesita acceso al total de todos los salarios de la tabla STAFF, podría crear una aplicación de SQL estático para ejecutar una sentencia SELECT SUM(SALARY) FROM STAFF, vincular la aplicación y otorgar el privilegio EXECUTE sobre el paquete de la aplicación a BUDGET. Esto permite a BUDGET obtener la información necesaria sin exponer la información que BUDGET no debería ver.

Conceptos relacionados:

- "Consideraciones sobre autorización para SQL dinámico" en la página 51
- "Consideraciones sobre autorización para SQL estático" en la página 52
- "Consideraciones sobre autorización para API" en la página 53
- "Authorization" en la publicación *Administration Guide: Planning*

Consideraciones sobre autorización para SQL dinámico

Para utilizar SQL dinámico en un paquete vinculado con DYNAMICRULES RUN (valor por omisión), la persona que ejecuta una aplicación de SQL dinámico debe tener los privilegios necesarios para emitir cada petición de SQL realizada, así como el privilegio EXECUTE sobre el paquete. Los privilegios se pueden otorgar al ID de autorización del usuario, a cualquier grupo del que el usuario sea miembro o a PUBLIC.

Si vincula la aplicación con la opción DYNAMICRULES BIND, DB2 asocia el ID de autorización con los paquetes de la aplicación. Esto permite que cualquier usuario que ejecute la aplicación herede los privilegios asociados con su ID de autorización.

Si el programa no contiene SQL estático, la persona que vincula la aplicación (para aplicaciones de SQL dinámico incorporado) sólo necesita la autorización

BINDADD sobre la base de datos. De nuevo, este privilegio se puede otorgar al ID de autorización del usuario, a un grupo del que el usuario sea miembro o a PUBLIC.

Cuando un programa muestra un comportamiento de vinculación o de definición, el usuario que ejecuta la aplicación sólo necesita el privilegio EXECUTE sobre el paquete para poderlo ejecutar. En el momento de la ejecución, el vinculador de un paquete que muestra un comportamiento de vinculación debe tener los privilegios necesarios para ejecutar todas las sentencias dinámicas generadas por el paquete, puesto que toda la comprobación de autorizaciones para sentencias dinámicas se lleva a cabo utilizando el ID del vinculador y no el de los ejecutores. Paralelamente, el que define una rutina cuyos paquetes muestran comportamiento de definición deben tener todos los privilegios necesarios para ejecutar todas las sentencias dinámicas generadas por el paquete con comportamiento de definición. Si tiene autorización SYSADM o DBADM y crea un paquete con comportamiento de vinculación, considere la posibilidad de utilizar la opción OWNER BIND para designar un ID de autorización diferente. La opción OWNER BIND evita que un paquete herede automáticamente los privilegios SYSADM o DBADM dentro de sentencias de SQL dinámico. Para obtener más información sobre las opciones de vinculación DYNAMICRULES y OWNER, consulte el mandato BIND. Para obtener más información sobre comportamientos de paquetes, consulte la descripción de los efectos de DYNAMICRULES en sentencias de SQL dinámico.

Conceptos relacionados:

- "Consideraciones sobre autorización para SQL incorporado" en la página 50
- "Consideraciones sobre autorización para SQL estático" en la página 52
- "Consideraciones sobre autorización para API" en la página 53
- "Autorizaciones y vinculación de rutinas que contienen SQL" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*

Información relacionada:

- "Mandato BIND" en la publicación *Consulta de mandatos*

Consideraciones sobre autorización para SQL estático

Para utilizar SQL estático, el usuario que ejecuta la aplicación sólo necesita el privilegio EXECUTE sobre el paquete. No se necesitan privilegios para cada una de las sentencias que forman el paquete. El privilegio EXECUTE se puede otorgar al ID de autorización del usuario, a cualquier grupo del que el usuario sea miembro o a PUBLIC.

A no ser que especifique la opción VALIDATE RUN cuando vincule la aplicación, el ID de autorización que utilice para vincular la aplicación debe tener los privilegios necesarios para llevar a cabo todas las sentencias de la aplicación. Si se especificó VALIDATE RUN en el momento de la vinculación (BIND), todos los errores de autorización correspondientes a cualquier SQL estático dentro de este paquete no harán que la operación BIND falle y dichas sentencias se volverán a validar en el momento de la ejecución. La persona que vincula la aplicación siempre debe tener autorización BINDADD. Los privilegios necesarios para ejecutar las sentencias se deben otorgar al ID de autorización del usuario o a PUBLIC. No se utilizan privilegios de grupo cuando se vinculan sentencias de SQL estático. Al igual que sucede con SQL dinámico, el privilegio BINDADD se puede otorgar al ID de autorización del usuario, a un grupo del que el usuario sea miembro o a PUBLIC.

Estas propiedades de SQL estático le proporcionan un control preciso sobre el acceso a información en DB2®.

Conceptos relacionados:

- "Consideraciones sobre autorización para SQL incorporado" en la página 50
- "Consideraciones sobre autorización para SQL dinámico" en la página 51
- "Consideraciones sobre autorización para API" en la página 53

Información relacionada:

- "Mandato BIND" en la publicación *Consulta de mandatos*

Consideraciones sobre autorización para API

La mayoría de las API que proporciona DB2® no necesitan el uso de privilegios, aunque muchas necesitan algún tipo de autorización para invocarlas. Para las API que necesitan privilegio, éste se debe otorgar al usuario que ejecuta la aplicación. El privilegio se puede otorgar al ID de autorización del usuario, a cualquier grupo del que el usuario sea miembro o a PUBLIC. Para obtener información sobre el privilegio y autorización necesarios para emitir cada llamada de API, consulte la descripción de la API.

Se puede acceder a algunas API a través de una interfaz de procedimiento almacenado. Para obtener información sobre si se puede acceder a una API específica a través de un procedimiento almacenado, consulte la descripción de dicha API.

Conceptos relacionados:

- "Consideraciones sobre autorización para SQL incorporado" en la página 50
- "Consideraciones sobre autorización para SQL dinámico" en la página 51
- "Consideraciones sobre autorización para SQL estático" en la página 52

Prueba de la aplicación

Las secciones siguientes describen cómo configurar un entorno de prueba y cómo depurar y optimizar la aplicación.

Configuración del entorno de prueba para una aplicación

Las secciones siguientes describen cómo configurar el entorno de prueba para la aplicación.

Configuración de un entorno de prueba

Para validar la aplicación, debe configurar un entorno de prueba. Por ejemplo, necesita una base de datos para probar el código SQL de la aplicación.

Procedimiento:

Para configurar el entorno de prueba, siga los siguientes pasos:

1. Cree una base de datos de prueba.

   Para crear una base de datos de prueba, escriba una pequeña aplicación de servidor que llame a la API CREATE DATABASE o utilice el procesador de línea de mandatos.
2. Cree tablas y vistas de prueba.

Si la aplicación actualiza, inserta o suprime datos de tablas y vistas, utilice datos de prueba para verificar su ejecución. Si la aplicación sólo recupera datos de tablas y vistas, considere la posibilidad de utilizar datos de nivel de producción cuando la pruebe.

3. Genere datos de prueba para las tablas.

   Los datos de entrada que se utilizan para probar una aplicación deben ser datos válidos que representen todas las posibles condiciones de entrada. Si la aplicación verifica que los datos de entrada son válidos, incluya datos válidos y no válidos para verificar que los datos válidos se procesan y los datos no válidos se marcan.
4. Depure y optimice la aplicación.

Tareas relacionadas:

- "Creación de tablas y vistas de prueba" en la página 54
- "Generación de datos de prueba" en la página 55
- "Depuración y optimización de una aplicación" en la página 57

Información relacionada:

- "sqlecrea - Create Database" en la publicación *Administrative API Reference*
- "Mandato CREATE DATABASE" en la publicación *Consulta de mandatos*

Creación de tablas y vistas de prueba

Para diseñar las tablas de prueba y vistas necesarias, primero analice los requisitos de datos de la aplicación. Para crear una tabla, necesita la autorización CREATETAB y el privilegio CREATEIN sobre el esquema. Consulte la sentencia CREATE TABLE para ver autorizaciones alternativas.

Procedimiento:

Para crear tablas de prueba:

1. Liste los datos a los que accede la aplicación y describa el modo en que se accede a cada elemento de datos. Por ejemplo, supongamos que la aplicación que se está desarrollando accede a las tablas TEST.TEMPL, TEST.TDEPT y TEST.TPROJ. Podría registrar el tipo de accesos, tal como se muestra en la siguiente tabla.

Tabla 1. Descripción de los datos de la aplicación

Nombre de tabla o vista	Insertar filas	Suprimir filas	Nombre columna	Tipo de datos	Acceso de actualización
TEST.TEMPL	No	No	EMPNO LASTNAME WORKDEPT PHONENO JOBCODE	CHAR(6) VARCHAR(15) CHAR(3) CHAR(4) DECIMAL(3)	Sí Sí Sí
TEST.TDEPT	No	No	DEPTNO MGRNO	CHAR(3) CHAR(6)	

Tabla 1. Descripción de los datos de la aplicación (continuación)

Nombre de tabla o vista	Insertar filas	Suprimir filas	Nombre columna	Tipo de datos	Acceso de actualización
TEST.TPROJ	Sí	Sí	PROJNO	CHAR(6)	Sí
			DEPTNO	CHAR(3)	Sí
			RESPEMP	CHAR(6)	Sí
			PRSTAFF	DECIMAL(5,2)	Sí
			PRSTDATE	DECIMAL(6)	Sí
			PRENDATE	DECIMAL(6)	

2. Cuando la descripción del acceso a datos de la aplicación esté completa, construya las tablas y vistas de prueba necesarias para probar la aplicación:
   - Cree una tabla de prueba cuando la aplicación modifique datos en una tabla o vista. Cree las siguientes tablas de prueba utilizando la sentencia de SQL CREATE TABLE:
     – TEMPL
     – TPROJ
   - Cree una vista de prueba cuando la aplicación no modifique datos en la base de datos de producción.

     En este ejemplo, cree una vista de prueba de la tabla TDEPT utilizando la sentencia de SQL CREATE VIEW.
3. Genere datos de prueba para las tablas.

Si el esquema de base de datos se está desarrollando junto con la aplicación, las definiciones de las tablas de prueba se pueden definir con más precisión repetidamente durante el proceso de desarrollo. Generalmente, la aplicación primaria no puede crear las tablas y acceder a las mismas porque el gestor de bases de datos no puede vincular sentencias que hacen referencia a tablas y vistas que no existen. Para que el proceso de creación y cambio de tablas no le lleve tanto tiempo, considere la posibilidad de desarrollar una aplicación separada para crear las tablas. También puede crear tablas de prueba de forma interactiva mediante el procesador de línea de mandatos (CLP).

Una vez finalizado el procedimiento, debe crear los temas relacionados para esta tarea.

Tareas relacionadas:

- "Generación de datos de prueba" en la página 55

Información relacionada:

- "Sentencia CREATE TABLE" en la publicación *Consulta de SQL, Volumen 2*

Generación de datos de prueba

Después de crear las tablas de prueba, puede llenarlas con datos de prueba para verificar el comportamiento de manejo de datos de la aplicación.

Procedimiento:

Utilice cualquiera de los siguientes métodos para insertar datos en una tabla:

- INSERT...VALUES (una sentencia de SQL) coloca una o más filas en una tabla cada vez que se emite el mandato.

- INSERT...SELECT obtiene datos de una tabla existente (basada en una cláusula SELECT) y los coloca en la tabla identificada con la sentencia INSERT.
- El programa de utilidad IMPORT o LOAD inserta gran cantidad de datos nuevos o existentes procedentes de una fuente definida.
- El programa de utilidad RESTORE se puede utilizar para duplicar el contenido de una base de datos existente en una base de datos de prueba idéntica utilizando una copia de seguridad (BACKUP) de la base de datos original.
- El programa de utilidad DB2MOVE mueve un gran número de tablas entre bases de datos DB2 situadas en estaciones de trabajo.

Las siguientes sentencias de SQL muestran una técnica que puede utilizar para llenar tablas con datos de prueba generados de forma aleatoria. Supongamos que la tabla EMP contiene cuatro columnas, ENO (número de empleado), LASTNAME (apellido), HIREDATE (fecha de contratación) y SALARY (salario del empleado) tal como se muestra en la siguiente sentencia CREATE TABLE:

```
CREATE TABLE EMP (ENO INTEGER, LASTNAME VARCHAR(30),
                  HIREDATE DATE, SALARY INTEGER);
```

Supongamos que desea llenar esta tabla con números de empleado, desde 1 a un número, por ejemplo 100, con datos aleatorios para el resto de las columnas. Puede hacerlo utilizando la siguiente sentencia de SQL:

```
INSERT INTO EMP
-- generar 100 registros
WITH DT(ENO) AS (VALUES(1) UNION ALL
SELECT ENO+1 FROM DT WHERE ENO < 100 ) 1

-- Utilice los registros generados en DT para crear otras columnas
-- del registro de empleado.
SELECT ENO, 2
    TRANSLATE(CHAR(INTEGER(RAND()*1000000)), 3
              CASE MOD(ENO,4) WHEN 0 THEN 'aeiou' || 'bcdfg'
                              WHEN 1 THEN 'aeiou' || 'hjklm'
                              WHEN 2 THEN 'aeiou' || 'npqrs'
                                     ELSE 'aeiou' || 'twxyz' END,
                                          '1234567890') AS LASTNAME,
    CURRENT DATE - (RAND()*10957) DAYS AS HIREDATE, 4
    INTEGER(10000+RAND()*200000) AS SALARY 5
  FROM DT;

SELECT * FROM EMP;
```

A continuación se explica la sentencia anterior:

1. La primera parte de la sentencia INSERT genera 100 registros correspondientes a los 100 primeros empleados utilizando una subconsulta recursiva para generar los números de empleado. Cada registro contiene el número de empleado. Para cambiar el número de empleados, utilice un número distinto de 100.
2. La sentencia SELECT genera la columna LASTNAME. Empieza generando un entero aleatorio de hasta 6 dígitos de longitud utilizando la función RAND. Luego convierte el entero en su formato de carácter numérico utilizando la función CHAR.
3. Para convertir los caracteres numéricos en caracteres alfabéticos, la sentencia utiliza la función TRANSLATE para convertir los diez caracteres numéricos (de 0 a 9) en caracteres alfabéticos. Puesto que hay más de 10 caracteres alfabéticos, la sentencia selecciona entre cinco conversiones diferentes. Esto da como resultado nombres que tienen suficientes vocales aleatorias para que se puedan pronunciar y por eso las vocales se incluyen en cada conversión.

4. La sentencia genera un valor de HIREDATE aleatorio. El valor de HIREDATE va, hacia atrás, desde la fecha actual hasta hace 30 años. HIREDATE se calcula restando un número aleatorio de días, comprendido entre 0 y 10.957, de la fecha actual. (10.957 es el número de días que hay en 30 años.)
5. Finalmente, la sentencia genera aleatoriamente el valor de SALARY. El salario mínimo es 10.000, al que se suma un número aleatorio comprendido entre 0 y 200.000.

Es posible que también desee considerar la posibilidad de crear prototipos de funciones definidas por el usuario (UDF) que desarrolle contra los datos de prueba.

Conceptos relacionados:

- "Import Overview" en la publicación *Data Movement Utilities Guide and Reference*
- "Load Overview" en la publicación *Data Movement Utilities Guide and Reference*
- "Métodos y funciones definidas por el usuario de DB2" en la página 20

Tareas relacionadas:

- "Depuración y optimización de una aplicación" en la página 57

Información relacionada:

- "Función escalar INSERT" en la publicación *Consulta de SQL, Volumen 1*
- "Mandato RESTORE DATABASE" en la publicación *Consulta de mandatos*

Depuración y optimización de una aplicación

Puede depurar y optimizar la aplicación mientras la desarrolla.

Procedimiento:

Para depurar y optimizar la aplicación:

- Cree prototipos de las sentencias de SQL. Puede utilizar el procesador de línea de mandatos, el recurso Explain, analizar las vistas del catálogo del sistema para obtener información sobre las tablas y bases de datos que manipula el programa y actualizar determinadas estadísticas del catálogo del sistema para simular condiciones de producción.
- Utilice el recurso del marcador para comprobar la sintaxis de las sentencias de SQL en aplicaciones que se desarrollan para DB2 Universal Database para z/OS y OS/390 o para ver el cumplimiento con el estándar de nivel básico SQL92. Este recurso se invoca durante la precompilación.
- Utilice las API de manejo de errores. Por ejemplo, puede utilizar las API de manejo de errores para imprimir todos los mensajes durante la fase de prueba.
- Utilice el supervisor del sistema de bases de datos para capturar determinada información sobre optimización para analizarla.

Conceptos relacionados:

- "Catalog statistics for modeling and what-if planning" en la publicación *Administration Guide: Performance*
- "Recursos para crear prototipos de sentencias de SQL" en la página 42
- "The database system monitor information" en la publicación *Administration Guide: Performance*

- "Requisitos del archivo fuente para aplicaciones de SQL incorporado" en la página 69

Parte 2. SQL incorporado

Capítulo 3. Visión general sobre SQL incorporado

Incorporación de sentencias de SQL en un lenguaje principal 61
Creación y preparación del archivo fuente 63
Paquetes, vinculación y SQL incorporado 65
Creación de paquetes para SQL incorporado . . 65
Precompilación de archivos fuente que contienen SQL incorporado 67
Requisitos del archivo fuente para aplicaciones de SQL incorporado 69
Compilación y enlace de archivos fuente que contienen SQL incorporado 70
Creación de paquetes mediante el mandato BIND 71
Creación de versiones de paquetes 72
Efecto de registros especiales en SQL dinámico vinculado 73
El registro especial CURRENT PACKAGE PATH para esquemas de paquete 73
Resolución de nombres de tabla no calificados. . 76
Consideraciones adicionales cuando se vincula 77
Ventajas de la vinculación diferida. 78
Contenido del archivo de vinculación. 78
Relaciones entre aplicación, archivo de vinculación y paquete 79
Indicaciones horarias generadas por el precompilador 79
Revinculación de paquetes 81

Incorporación de sentencias de SQL en un lenguaje principal

Puede escribir aplicaciones con sentencias de SQL incorporadas dentro de un lenguaje principal. Las sentencias de SQL proporcionan la interfaz de base de datos, mientras que el lenguaje principal proporciona el soporte restante necesario para que se ejecute la aplicación.

Procedimiento:

Utilice los ejemplos de la tabla siguiente como guía para ver cómo se incorporan sentencias de SQL en una aplicación de lenguaje principal. En el ejemplo, la aplicación comprueba el campo SQLCODE de la estructura SQLCA para determinar si la actualización ha resultado satisfactoria.

Tabla 2. Incorporación de sentencias de SQL en un lenguaje principal

Lenguaje	Código fuente de ejemplo

C/C++	`EXEC SQL UPDATE staff SET job = 'Clerk' WHERE job = 'Mgr';`
	`if ( SQLCODE < 0 )`
	`  printf( "Update Error:  SQLCODE = %ld \n", SQLCODE );`

Java (SQLJ)	`try {`
	`  #sql { UPDATE staff SET job = 'Clerk' WHERE job = 'Mgr' };`
	`}`
	`catch (SQLException e) {`
	`  println( "Update Error:  SQLCODE = " + e.getErrorCode() );`
	`}`

COBOL	`EXEC SQL UPDATE staff SET job = 'Clerk' WHERE job = 'Mgr' END_EXEC.`
	`IF SQLCODE LESS THAN 0`
	`  DISPLAY 'UPDATE ERROR:  SQLCODE = ', SQLCODE.`

FORTRAN	`EXEC SQL UPDATE staff SET job = 'Clerk' WHERE job = 'Mgr'`
	`if ( sqlcode .lt. 0 ) THEN`
	`  write(*,*) 'Update error:  sqlcode = ', sqlcode`

Las sentencias de SQL que se colocan en una aplicación no son específicas del lenguaje principal. El gestor de bases de datos proporciona una forma de convertir la sintaxis de SQL para que la procese el lenguaje de sistema principal:

- Para los lenguajes C, C++, COBOL o FORTRAN, esta conversión la maneja el precompilador de DB2. El precompilador de DB2 se invoca utilizando el

mandato PREP. El precompilador convierte las sentencias de SQL incorporado directamente en llamadas a API de servicios de tiempo de ejecución de DB2.

- Para el lenguaje Java, el conversor SQLJ convierte cláusulas SQLJ en sentencias JDBC. El conversor SQLJ se invoca con el mandato **sqlj**.

Cuando el precompilador procesa un archivo fuente, busca específicamente sentencias de SQL y evita el lenguaje principal que no es SQL. Puede encontrar sentencias de SQL porque están especificadas entre delimitadores especiales. Los ejemplos de la tabla siguiente muestran cómo utilizar delimitadores y comentarios para crear sentencias de SQL incorporado en los lenguajes de sistema principal compilados soportados.

Tabla 3. Incorporación de sentencias de SQL en un lenguaje principal

Lenguaje	Código fuente de ejemplo

C/C++	`/* Aquí sólo se permiten comentarios de C o C++ */`
	`EXEC SQL`
	`-- Aquí se permiten comentarios de SQL`
	`/* o comentarios de C */`
	`// o comentarios de C++`
	`DECLARE C1 CURSOR FOR sname;`
	`/* Aquí sólo se permiten comentarios de C o C++ */`

SQLJ	`/* Aquí sólo se permiten comentarios de Java */`
	`#sql c1 = {`
	`-- Aquí se permiten comentarios de SQL`
	`/* comentarios de Java o */`
	`// comentarios de Java`
	`SELECT name FROM employee`
	`};`
	`/* Aquí sólo se permiten comentarios de Java */`

COBOL	`* Consulte la documentación de COBOL para conocer`
	`* las reglas sobre comentarios`
	`* Aquí sólo se permiten comentarios de COBOL`
	`EXEC SQL`
	`-- Aquí se permiten comentarios de SQL`
	`* o comentarios de COBOL de línea completa`
	`DECLARE C1 CURSOR FOR sname END-EXEC.`
	`* Aquí sólo se permiten comentarios de COBOL`

FORTRAN	`C Aquí sólo se permiten comentarios de FORTRAN`
	`EXEC SQL`
	`+ -- comentarios de SQL y`
	`C comentarios de FORTRAN de línea completa`
	`+ DECLARE C1 CURSOR FOR sname`
	`I=7 ! Fin de línea Se permiten comentarios de FORTRAN`
	`C Aquí sólo se permiten comentarios de FORTRAN`

Conceptos relacionados:

- "SQL incorporado en aplicaciones REXX" en la página 531
- "Sentencias de SQL incorporado en C y C++" en la página 150
- "Sentencias de SQL incorporado en COBOL" en la página 196
- "Sentencias de SQL incorporado en FORTRAN" en la página 219

Creación y preparación del archivo fuente

Puede crear el código fuente en un archivo ASCII estándar, denominado un archivo fuente, utilizando un editor de texto. El archivo fuente debe tener la extensión adecuada para el lenguaje principal en el que escribe el código.

Nota: No todas las plataformas dan soporte a todos los lenguajes principales.

Para este tema, daremos por supuesto que ya ha escrito el código fuente.

Si ha escrito la aplicación utilizando un lenguaje principal compilado, debe seguir pasos adicionales para crear la aplicación. Además de compilar y enlazar el programa, debe *precompilarlo* y *vincularlo*.

Dicho de forma sencilla, la precompilación convierte sentencias de SQL incorporado en llamadas a API en tiempo de ejecución de DB2 que un compilador de sistema principal puede procesar y crea un archivo de vinculación. El archivo de vinculación contiene información sobre las sentencias de SQL del programa de aplicación. El mandato `BIND` crea un *paquete* en la base de datos. Opcionalmente, el precompilador puede llevar a cabo al paso de vinculación en el momento de la precompilación.

La vinculación es el proceso de crear un *paquete* a partir de un archivo de vinculación y de almacenarlo en una base de datos. Si la aplicación accede a más de una base de datos, debe crear un paquete para cada base de datos.

La figura siguiente muestra el orden de estos pasos, junto con los distintos módulos de una típica aplicación de DB2 compilada.

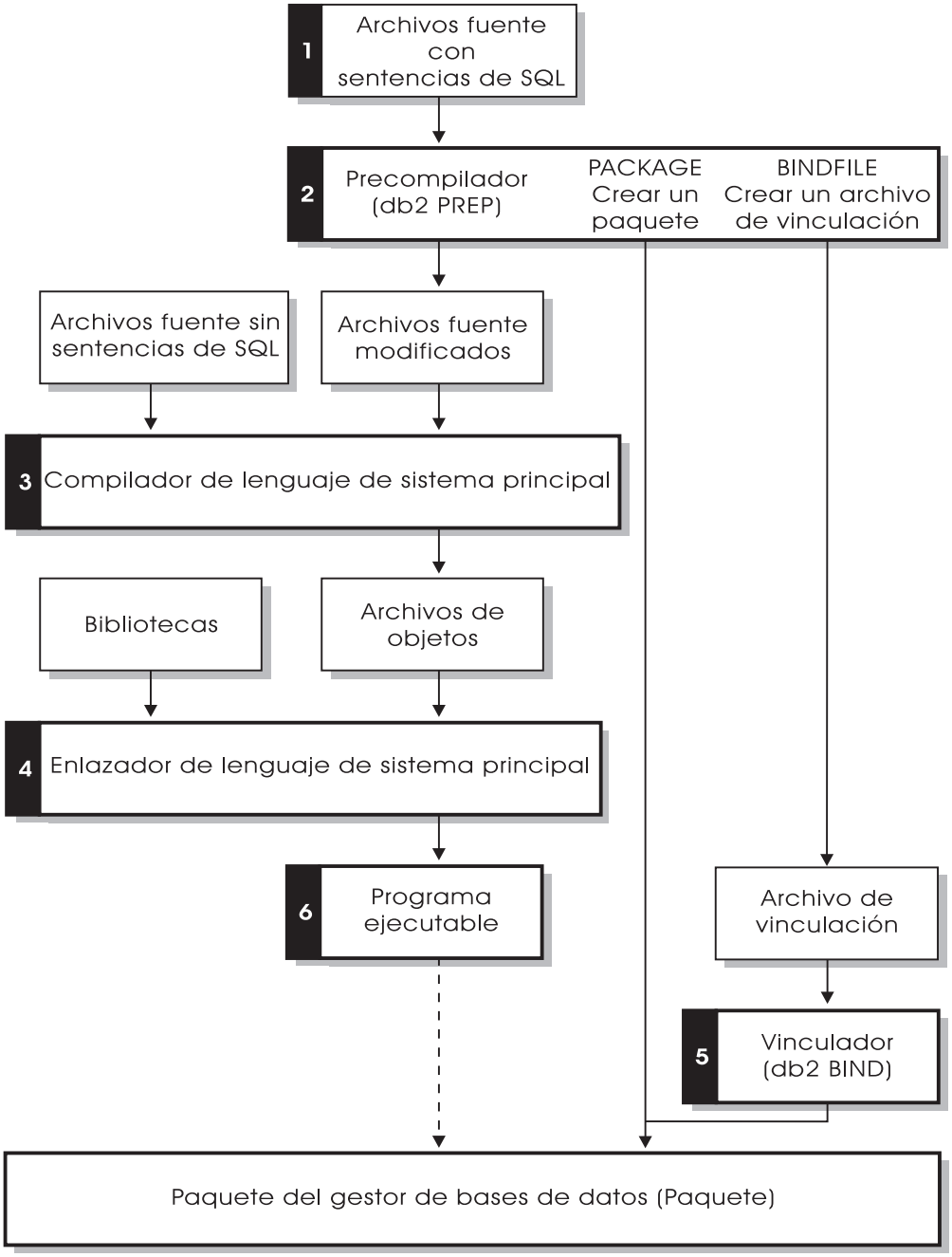


Figura 1. Preparación de programas escritos en lenguajes principal compilados

Conceptos relacionados:

- "Precompilación de archivos fuente que contienen SQL incorporado" en la página 67
- "Requisitos del archivo fuente para aplicaciones de SQL incorporado" en la página 69
- "Compilación y enlace de archivos fuente que contienen SQL incorporado" en la página 70
- "SQL incorporado" en la página 8

Información relacionada:

- "Mandato BIND" en la publicación *Consulta de mandatos*

Paquetes, vinculación y SQL incorporado

Las secciones siguientes describen cómo crear paquetes para aplicaciones de SQL incorporado, así como otros temas, como la vinculación diferida y las relaciones entre la aplicación, el archivo de vinculación y el paquete.

Creación de paquetes para SQL incorporado

Para ejecutar aplicaciones escritas en lenguajes principales compilados, debe crear los paquetes que necesita el gestor de bases de datos en el momento de la ejecución. Esto implica llevar a cabo los pasos siguientes, tal como se muestra en la siguiente figura:

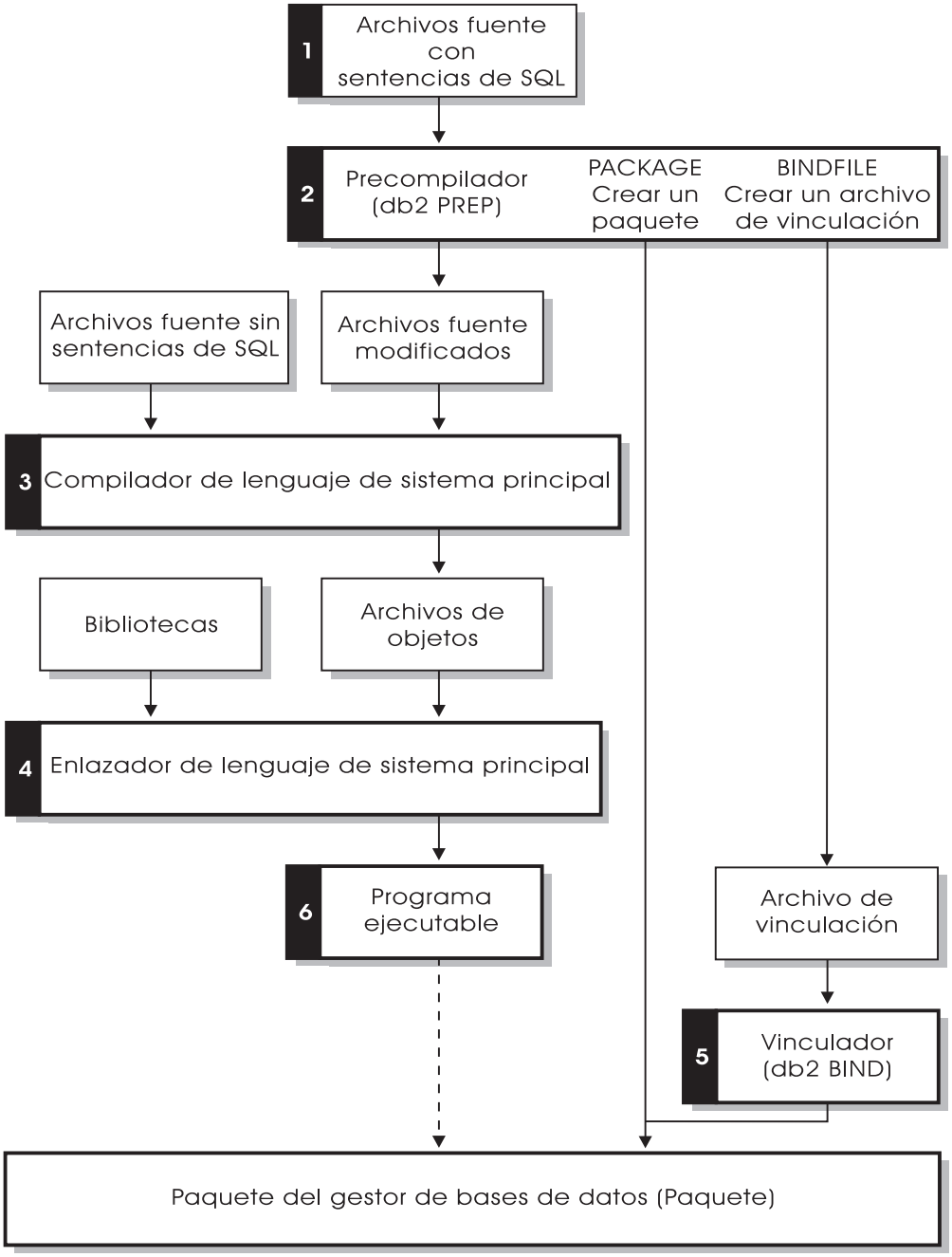


Figura 2. Preparación de programas escritos en lenguajes principales compilados

- Precompilación (paso 2), para convertir las sentencias fuente de SQL incorporado en un formato que el gestor de bases de datos pueda utilizar
- Compilación y enlace (pasos 3 y 4), para crear los módulos de objetos necesarios, y
- Vinculación (paso 5), para crear el paquete que utilizará el gestor de bases de datos cuando se ejecute el programa.

Conceptos relacionados:

- "Precompilación de archivos fuente que contienen SQL incorporado" en la página 67

- "Requisitos del archivo fuente para aplicaciones de SQL incorporado" en la página 69
- "Compilación y enlace de archivos fuente que contienen SQL incorporado" en la página 70
- "Creación de paquetes mediante el mandato BIND" en la página 71
- "Creación de versiones de paquetes" en la página 72
- "Efecto de registros especiales en SQL dinámico vinculado" en la página 73
- "Resolución de nombres de tabla no calificados" en la página 76
- "Consideraciones adicionales cuando se vincula" en la página 77
- "Ventajas de la vinculación diferida" en la página 78
- "Relaciones entre aplicación, archivo de vinculación y paquete" en la página 79
- "Indicaciones horarias generadas por el precompilador" en la página 79
- "Revinculación de paquetes" en la página 81

Información relacionada:

- "db2bfd - Mandato Herramienta de descripción de archivo de vinculación" en la publicación *Consulta de mandatos*

Precompilación de archivos fuente que contienen SQL incorporado

Después de crear los archivos fuente, debe precompilar cada archivo de lenguaje principal que contenga sentencias de SQL con el mandato `PREP` para archivos fuente de lenguaje principal. El precompilador convierte las sentencias de SQL contenidas en el archivo fuente en comentarios y genera las llamadas a API en tiempo de ejecución de DB2 para dichas sentencias.

Antes de precompilar una aplicación, debe conectarse con el servidor, de forma implícita o explícita. Aunque precompile programas de aplicación en la estación de trabajo cliente y el precompilador genere fuente modificada y mensajes en el cliente, el precompilador utiliza la conexión con el servidor para llevar a cabo parte de la validación.

El precompilador también crea la información que necesita el gestor de bases de datos para procesar las sentencias de SQL contra una base de datos. Esta información se almacena en un paquete, en un archivo de vinculación o en ambos, en función de las opciones del precompilador seleccionadas.

A continuación se muestra un ejemplo típico de cómo utilizar el precompilador. Para precompilar un archivo fuente de SQL incorporado C denominado *filename.sqc*, puede emitir el siguiente mandato para crear un archivo fuente C con el nombre por omisión `filename.c` y un archivo de vinculación con el nombre por omisión `filename.bnd`:

```
DB2 PREP filename.sqc BINDFILE
```

El precompilador genera un máximo de cuatro tipos de salida:

Fuente modificada

Este archivo es la nueva versión del archivo fuente original después de que el precompilador convierta las sentencias de SQL en llamadas a API en tiempo de ejecución de DB2. Se le asigna la extensión adecuada para el lenguaje principal.

Paquete

Si utiliza la opción PACKAGE (el valor por omisión) o si no especifica ninguna de las opciones BINDFILE, SYNTAX o SQLFLAG, el paquete se almacena en la base de datos conectada. El paquete contiene toda la información necesaria para ejecutar las sentencias de SQL estático de un archivo fuente particular contra esta base de datos únicamente. A no ser que especifique otro nombre con la opción PACKAGE USING, el precompilador forma el nombre del paquete a partir de los 8 primeros caracteres del nombre del archivo fuente.

Si utiliza la opción PACKAGE sin SQLERROR CONTINUE, la base de datos que se utiliza durante el proceso de precompilación debe contener todos los objetos de base de datos a los que hacen referencia las sentencias de SQL estático en el archivo fuente. Por ejemplo, no puede precompilar una sentencia SELECT a no ser que la tabla a la que hace referencia exista en la base de datos.

Con la opción VERSION, el archivo de vinculación (si se utiliza la opción BINDFILE) y el paquete (tanto si se vincula en el momento de PREP como si se vincula por separado) se designarán con un identificador de versión particular. Pueden existir simultáneamente muchas versiones de paquetes con el mismo nombre y creador.

Archivo de vinculación

Si utiliza la opción BINDFILE, el precompilador crea un archivo de vinculación (con la extensión `.bnd`) que contiene los datos necesarios para crear un paquete. Este archivo se puede utilizar más adelante con el mandato `BIND` para vincular la aplicación a una o más bases de datos. Si especifica BINDFILE y no especifica la opción PACKAGE, la vinculación se difiere hasta que invoca el mandato `BIND`. Observe que para el procesador de línea de mandatos (CLP), el valor por omisión para `PREP` no especifica la opción BINDFILE. Por lo tanto, si utiliza el CLP y desea que la vinculación se difiera, tiene que especificar la opción BINDFILE.

Si se especifica SQLERROR CONTINUE se crea un paquete, aunque se produzcan errores al vincular sentencias de SQL. Estas sentencias que no se pueden vincular por motivos de autorización o de existencia se pueden vincular de forma incremental en el momento de la ejecución si también se especifica VALIDATE RUN. Si se intenta ejecutarlas en el momento de la ejecución se genera un error.

Archivo de mensajes

Si utiliza la opción MESSAGES, el precompilador redirige los mensajes al archivo indicado. Estos mensajes incluyen avisos y mensajes de error que describen problemas encontrados durante la precompilación. Si el archivo fuente no se precompila satisfactoriamente, utilice los mensajes de aviso y de error para determinar el problema, corrija el archivo fuente y luego vuelva a intentar precompilar el archivo fuente. Si no utiliza la opción MESSAGES, los mensajes de precompilación se graban en la salida estándar.

Conceptos relacionados:

• "Creación de versiones de paquetes" en la página 72

Información relacionada:

- "Mandato PRECOMPILE" en la publicación *Consulta de mandatos*

Requisitos del archivo fuente para aplicaciones de SQL incorporado

Siempre debe precompilar un archivo fuente contra una base de datos específica, incluso si finalmente no utiliza la base de datos con la aplicación. En la práctica, puede utilizar una base de datos de prueba para desarrollo y, después de probar por completo la aplicación, puede vincular su archivo de vinculación a una o más bases de datos de producción. Esta práctica recibe el nombre de *vinculación diferida*.

Si la aplicación utiliza una página de códigos distinta de la página de códigos de la base de datos, debe tener en cuenta qué página de códigos debe utilizar al precompilar.

Si la aplicación utiliza funciones definidas por el usuario (UDF) o tipos diferenciados definidos por el usuario (UDT), es posible que tenga que utilizar la opción FUNCPATH al precompilar la aplicación. Esta opción especifica la vía de acceso de función que se utiliza para resolver las UDF y UDT para aplicaciones que contienen SQL estático. Si no se especifica FUNCPATH, la vía de acceso de función por omisión es *SYSIBM*, *SYSFUN*, *USER*, donde *USER* hace referencia al ID de usuario actual.

Para precompilar un programa de aplicación que accede a más de un servidor, puede hacer una de las siguientes cosas:

- Dividir las sentencias de SQL correspondientes a cada base de datos en archivos fuente separados. No combine sentencias de SQL correspondientes a distintas bases de datos en el mismo archivo. Cada archivo fuente se puede precompilar contra la base de datos apropiada. Este es el método recomendado.
- Codificar la aplicación utilizando únicamente sentencias de SQL dinámico y vincular contra cada base de datos a la que va a acceder el programa.
- Si todas las bases de datos se parecen, es decir, tienen la misma definición, puede agrupar las sentencias de SQL en un archivo fuente.

Los mismos procedimientos se aplican si la aplicación va a acceder a un servidor de aplicaciones de sistema principal, AS/400® o iSeries a través de DB2 Connect. Precompílela contra el servidor al que se va a conectar, utilizando las opciones de `PREP` disponibles para dicho servidor.

Si va a precompilar una aplicación que se va a ejecutar en DB2 Universal Database para z/OS y OS/390, considere la posibilidad de utilizar el recurso del marcador para comprobar la sintaxis de las sentencias de SQL. El marcador indica la sintaxis de SQL a la que da soporte DB2 Universal Database pero a la que no da soporte DB2 Universal Database para z/OS y OS/390. También puede utilizar el marcador para comprobar que la sintaxis de SQL cumple con la sintaxis de nivel básico de SQL92. Puede utilizar la opción SQLFLAG en el mandato `PREP` para invocarlo y para especificar la versión de la sintaxis de SQL de DB2 Universal Database para z/OS y OS/390 que se va a utilizar para la comparación. El recurso del marcador no obliga a realizar ningún cambio en el uso de SQL; sólo emite mensajes de información y de aviso sobre las incompatibilidades de la sintaxis y no termina el preproceso de forma anómala.

Conceptos relacionados:

- "Ventajas de la vinculación diferida" en la página 78

- "Conversión de caracteres entre páginas de códigos diferentes" en la página 650
- "Cuándo se produce conversión de la página de códigos" en la página 650
- "Sustitución de caracteres durante conversiones de páginas de códigos" en la página 651
- "Conversiones de páginas de códigos soportadas" en la página 652
- "Factor de expansión de conversión de página de códigos" en la página 653

Información relacionada:

- "Mandato PRECOMPILE" en la publicación *Consulta de mandatos*

Compilación y enlace de archivos fuente que contienen SQL incorporado

Compile los archivos fuente modificados y los archivos fuente adicionales que no contienen sentencias de SQL utilizando el compilador de lenguaje principal adecuado. El compilador de lenguaje convierte cada archivo fuente modificado en un *módulo de objeto*.

Consulte la documentación sobre programación correspondiente a su plataforma operativa para ver las excepciones a las opciones del compilador por omisión. Consulte la documentación del compilador para ver una descripción completa de las opciones disponibles del compilador.

El enlazador del lenguaje principal crea una aplicación ejecutable. Por ejemplo:

- En sistemas operativos Windows®, la aplicación puede ser un archivo ejecutable o una biblioteca de enlace dinámico (DLL).
- En los sistemas basados en UNIX®, la aplicación puede ser un módulo de carga ejecutable o una biblioteca compartida.

Nota: Aunque las aplicaciones pueden ser bibliotecas de enlace dinámico (DLL) en los sistemas operativos Windows, las DLL son cargadas directamente por la aplicación y no por el gestor de bases de datos de DB2®. En los sistemas operativos Windows, el gestor de bases de datos puede cargar las DLL. Los procedimientos almacenados se crean normalmente como DLL o bibliotecas compartidas.

Para crear el archivo ejecutable, enlace lo siguiente:

- Módulos de objetos de usuario, generados por el compilador del lenguaje a partir de los archivos fuente modificados y otros archivos que no contienen sentencias de SQL.
- API de biblioteca de lenguaje principal, que se suministran con el compilador del lenguaje.
- La biblioteca del gestor de bases de datos que contiene las API del gestor de bases de datos correspondientes a su entorno operativo. Consulte la documentación sobre programación adecuada para su plataforma operativa para ver el nombre específico de la biblioteca del gestor de bases de datos que necesita para las API del gestor de bases de datos.

Conceptos relacionados:

- "Procedimientos almacenados de DB2" en la página 19

Tareas relacionadas:

- "Creación y ejecución de aplicaciones REXX" en la página 541
- "Creación de applets JDBC" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de aplicaciones JDBC" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de applets SQLJ" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de aplicaciones SQLJ" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de aplicaciones C para UNIX" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de aplicaciones C++ para UNIX" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de aplicaciones IBM COBOL en AIX" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de aplicaciones Micro Focus COBOL para UNIX" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Creación de paquetes mediante el mandato BIND

La vinculación es el proceso que crea el paquete que necesita el gestor de bases de datos para poder acceder a la base de datos cuando se ejecuta la aplicación. La vinculación se puede realizar de forma implícita, especificando la opción PACKAGE durante la precompilación, o de forma explícita, utilizando el mandato `BIND` contra el archivo de vinculación creado durante la precompilación.

A continuación se muestra un ejemplo típico de cómo utilizar el mandato `BIND`. Para vincular un archivo de vinculación denominado *filename.bnd* a la base de datos, puede emitir el siguiente mandato:

```
DB2 BIND nombre_archivo.bnd
```

Se crea un paquete para cada módulo de código fuente precompilado por separado. Si una aplicación tiene cinco archivos fuente, de los cuales tres necesitan precompilación, se crean tres paquetes o archivos de vinculación. Por omisión, a cada paquete se le asigna un nombre que coincide con el nombre del módulo fuente a partir del cual se ha originado el archivo `.bnd`, pero truncado a 8 caracteres. Para especificar de forma explícita otro nombre de paquete, debe utilizar la opción PACKAGE USING en el mandato `PREP`. La versión de un paquete la proporciona la opción de precompilación VERSION y su valor por omisión es una serie vacía. Si el nombre y esquema de este paquete recién creado coinciden con el nombre de un paquete que existe actualmente en la base de datos de destino, pero el identificador de versión difiere, se crea un paquete nuevo y se conserva el paquete anterior. Sin embargo, si existe un paquete cuyo nombre, esquema y versión coinciden con los del paquete que se está vinculando, el paquete se elimina y se sustituye por el paquete nuevo que se está vinculando (si se especifica ACTION ADD en la vinculación se evita que se devuelva un error (SQL0719)).

Información relacionada:

- "Mandato BIND" en la publicación *Consulta de mandatos*
- "Mandato PRECOMPILE" en la publicación *Consulta de mandatos*

Creación de versiones de paquetes

Si tiene que crear varias versiones de una aplicación, puede utilizar la opción VERSION del mandato PRECOMPILE. Esta opción permite la coexistencia de varias versiones del mismo nombre de paquete (es decir, el nombre del paquete y el nombre del creador). Por ejemplo, supongamos que tiene una aplicación denominada `foo` que se compila desde `foo.sqc`. Precompilaría y vincularía el paquete `foo` a la base de datos y distribuiría la aplicación a los usuarios. Luego los usuarios podrían ejecutar la aplicación. Para realizar cambios en la aplicación, actualizaría `foo.sqc` y luego repetiría el proceso de recompilación, vinculación y envío de la aplicación a los usuarios. Si no se especifica la opción VERSION para la primera o segunda precompilación de `foo.sqc`, el primer paquete se sustituye por el segundo. Cualquier usuario que intente ejecutar la versión antigua de la aplicación recibirá un error SQLCODE -818, que indica un error de falta de coincidencia de indicación horaria.

Para evitar el error de falta de coincidencia de indicación horaria y para permitir que ambas versiones de la aplicación se ejecutan al mismo tiempo, utilice la creación de versiones de paquetes. Como ejemplo, cuando cree la primera versión de `foo`, precompílela utilizando la opción VERSION, del siguiente modo:

```
DB2 PREP FOO.SQC VERSION V1.1
```

Ahora se puede ejecutar esta primera versión del programa. Cuando cree la nueva versión de `foo`, precompílela con el mandato:

```
DB2 PREP FOO.SQC VERSION V1.2
```

En este momento esta nueva versión de la aplicación también se ejecutará, aunque aún se estén ejecutando instancias de la primera aplicación. Puesto que la versión de paquete correspondiente al primer paquete es V1.1 y la versión de paquete correspondiente al segundo es V1.2, no hay conflicto de nombres: ambos paquetes existirán en la base de datos y ambas versiones de la aplicación se pueden utilizar.

Puede utilizar la opción ACTION de los mandatos PRECOMPILE o BIND junto con la opción VERSION del mandato PRECOMPILE. La opción ACTION sirve para controlar el modo en que las distintas versiones de paquetes se pueden añadir o sustituir.

Los privilegios de paquetes no tienen granularidad a nivel de versión. Es decir, una acción GRANT o REVOKE de un privilegio de paquete se aplica a todas las versiones de un paquete que comparten el nombre y el creador. Así, si se otorgaran los privilegios del paquete `foo` a un usuario o un grupo después de que se creara la versión V1.1, cuando se distribuyera la versión V1.2 el usuario o grupo tendría los mismos privilegios sobre la versión V1.2. Este comportamiento suele ser necesario porque normalmente los mismos usuarios y grupos tienen los mismos privilegios sobre todas las versiones de un paquete. Si no desea que se apliquen los mismos privilegios de paquete a todas las versiones de una aplicación, no utilice la opción PRECOMPILE VERSION al realizar versiones del paquete. En su lugar, utilice distintos nombres de paquetes (cambiando el nombre del archivo fuente actualizado o utilizando la opción PACKAGE USING para cambiar el nombre del paquete de forma explícita).

Conceptos relacionados:

- "Indicaciones horarias generadas por el precompilador" en la página 79

Información relacionada:

- "Mandato BIND" en la publicación *Consulta de mandatos*
- "Mandato PRECOMPILE" en la publicación *Consulta de mandatos*

Efecto de registros especiales en SQL dinámico vinculado

Para sentencias preparadas de forma dinámica, los valores de varios registros especiales determinan el entorno de compilación de sentencias:

- El registro especial CURRENT QUERY OPTIMIZATION determina qué clase de optimización se utiliza.
- El registro especial CURRENT PATH determina la vía de acceso de función utilizada para la resolución de UDF y UDT.
- El registro CURRENT EXPLAIN SNAPSHOT determina qué información de instantánea de Explain se captura.
- El registro CURRENT EXPLAIN MODE determina si la información de tabla de Explain se captura, para cualquier sentencia de SQL dinámico que se pueda elegir. Los valores por omisión correspondientes a estos registros especiales son los mismos que se utilizan para las opciones de vinculación relacionadas.

Información relacionada:

- "Registro especial CURRENT EXPLAIN MODE" en la publicación *Consulta de SQL, Volumen 1*
- "Registro especial CURRENT EXPLAIN SNAPSHOT" en la publicación *Consulta de SQL, Volumen 1*
- "Registro especial CURRENT PATH" en la publicación *Consulta de SQL, Volumen 1*
- "Registro especial CURRENT QUERY OPTIMIZATION" en la publicación *Consulta de SQL, Volumen 1*

El registro especial CURRENT PACKAGE PATH para esquemas de paquete

Los esquemas de paquete proporcionan un método para la agrupación lógica de paquetes. Existen diferentes métodos para agrupar paquetes y formar esquemas. Algunas implementaciones utilizan un esquema por cada entorno (por ejemplo, un esquema de producción y un esquema de prueba). Otras implementaciones utilizan un esquema por cada área de negocio (por ejemplo, los esquemas stocktrd y onlinebnk) o un esquema por cada aplicación (por ejemplo, stocktrdAddUser y onlinebnkAddUser). También puede agrupar paquetes con finalidades de administración general o para proporcionar variaciones en los paquetes (por ejemplo, mantener variaciones de copia de seguridad de aplicaciones o probar variaciones nuevas de aplicaciones).

Cuando se utilizan varios esquemas para los paquetes, el gestor de la base de datos debe determinar el esquema en el que ha de buscarse un paquete. Para realizar esta tarea, el gestor de bases de datos utiliza el valor del registro especial CURRENT PACKAGESET. Puede definir este registro especial para un solo nombre de esquema para indicar que cualquier paquete a invocar pertenece a ese esquema. Si una aplicación utiliza paquetes en esquemas diferentes, es posible que tenga que emitirse una sentencia SET CURRENT PACKAGESET antes de que se invoque cualquier paquete en el caso de que el esquema para el paquete sea diferente respecto del paquete anterior.

Nota: Solo DB2® Universal Database para OS/390® y z/OS™ tiene un registro especial CURRENT PACKAGESET, que le permite definir explícitamente el

valor (un nombre de esquema individual) con la correspondiente sentencia SET CURRENT PACKAGESET. Aunque DB2 Universal Database™ para Linux, UNIX® y Windows® tiene una sentencia SET CURRENT PACKAGESET, carece de un registro especial CURRENT PACKAGESET. Esto significa que CURRENT PACKAGESET no se puede utilizar en otros contextos (tales como una sentencia SELECT) con DB2 Universal Database para Linux, UNIX y Windows. DB2 Universal Database para AS/00 no proporciona soporte para CURRENT PACKAGESET.

DB2 tiene más flexibilidad cuando puede tomar en consideración una lista de esquemas durante la resolución de paquetes. La lista de esquemas es similar a la vía de acceso a SQL que se proporciona por medio del registro especial CURRENT PATH. La lista de esquemas se utiliza para las funciones definidas por el usuario, procedimientos, métodos y tipos diferenciados.

Nota: La vía de acceso a SQL es una lista de nombres de esquema que DB2 debería tener en cuenta al intentar determinar el esquema para un nombre de tipo diferenciado, método, procedimiento o función no calificada.

Si necesita asociar múltiples variaciones de un paquete (es decir, múltiples conjuntos de opciones BIND de un paquete) con un único programa compilado, considere la posibilidad de aislar la vía de acceso de los esquemas que se utilizan para objetos de SQL respecto de la vía de acceso de los esquemas que se utilizan para paquetes.

El registro especial CURRENT PACKAGE PATH le permite especificar una lista de esquemas de paquete. Otros productos de la familia DB2 proporcionan una capacidad similar con registros especiales tales como CURRENT PATH y CURRENT PACKAGESET, que se utilizan para procedimientos anidados y funciones definidas por el usuario, sin corromper el entorno de ejecución de la aplicación invocante. El registro especial CURRENT PACKAGE PATH proporciona esta capacidad para la resolución de esquemas del paquete.

Muchas instalaciones utilizan más de un esquema para los paquetes. Si no especifica una lista de esquemas de paquete, deberá emitir la sentencia SET CURRENT PACKAGESET (que puede contener como máximo un nombre de esquema) cada vez que necesite un paquete de un esquema diferente. Sin embargo, si emite una sentencia SET CURRENT PACKAGE PATH al principio de la aplicación para especificar una lista de nombres de esquema, no necesita emitir una sentencia SET CURRENT PACKAGESET cada vez que se necesite un paquete en un esquema diferente. Esta posibilidad es especialmente útil si está creando una aplicación SQLJ, pues la aplicación puede examinar una lista de esquemas de paquete sin tener que emitir una sentencia SET CURRENT PACKAGESET cada vez que conmute entre SQLJ y JDBC.

Por ejemplo, suponga que existen los paquetes siguientes y, utilizando la lista siguiente, que desea invocar el primero que existe en el servidor: SCHEMA1.PKG1, SCHEMA2.PKG2, SCHEMA3.PKG3, SCHEMA4.PKG y SCHEMA5.PKG5. Suponiendo que existe el soporte actual para una sentencia SET CURRENT PACKAGESET de DB2 Universal Database para Linux, UNIX y Windows (es decir, aceptando un solo nombre de esquema), tendría que emitirse una sentencia SET CURRENT PACKAGESET antes de intentar invocar cada paquete para especificar el esquema específico. Para este ejemplo, se tendría que emitir cinco sentencias SET CURRENT PACKAGESET. Sin embargo, es suficiente la utilización del registro especial CURRENT PACKAGE PATH, una única sentencia SET. Por ejemplo:

```
SET CURRENT PACKAGE PATH = SCHEMA1, SCHEMA2, SCHEMA3, SCHEMA, SCHEMA5;
```

Nota: En DB2 Universal Database para Linux, UNIX y Windows, puede definir el registro especial CURRENT PACKAGE PATH en el archivo db2cli.ini, utilizando la API de SQLSetConnectAttr, en la estructura de SQLE-CLIENT-INFO e incluyendo la sentencia SET CURRENT PACKAGE PATH en programas de SQL intercalado. Solo DB2 Universal Database para OS/390 y z/OS, Versión 8 o posterior, da soporte a la sentencia SET CURRENT PACKAGE PATH. Si emite esta sentencia para un servidor de DB2 Universal Database para Linux, UNIX o Windows, o para DB2 Universal Database para AS/00, se devuelve -30005.

Puede utilizar múltiples esquemas para mantener diversas variaciones de un paquete. Estas variaciones pueden ser muy útiles para ayudar a controlar los cambios efectuados en entornos de producción. También puede utilizar diferentes variaciones de un paquete para conservar una versión de copia de seguridad de un paquete, o una versión de prueba de un paquete (por ejemplo, para evaluar el impacto de un índice nuevo). Se utiliza una versión anterior de un paquete del mismo modo que una aplicación de copia de seguridad (módulo de carga o ejecutable), específicamente para proporcionar la posibilidad de volver a una versión anterior.

Por ejemplo, suponga que el esquema PROD incluye los paquetes actuales utilizados por las aplicaciones de producción, y que el esquema BACKUP guarda una copia de seguridad de esos paquetes. Una versión nueva de la aplicación (y por tanto de los paquetes) se promociona a la versión de producción mediante la vinculación por medio del esquema PROD. Las copias de seguridad de los paquetes se crean vinculando la versión actual de las aplicaciones mediante el esquema de copia de seguridad (BACKUP). Después, durante la ejecución, puede utilizar la sentencia SET CURRENT PACKAGE PATH para especificar el orden en el que se deben examinar los esquemas para los paquetes. Suponga que se ha vinculado una copia de seguridad de la aplicación MYAPPL utilizando el esquema BACKUP y que la versión de la aplicación que está actualmente en producción se ha vinculado al esquema PROD creando un paquete PROD.MYAPPL. Para especificar que debe utilizarse la variación del paquete contenida en el esquema PROD si está disponible (de lo contrario se utilizará la variación contenida en el esquema BACKUP), emita la sentencia SET siguiente para el registro especial:

```
SET CURRENT PACKAGE PATH = PROD, BACKUP;
```

Si necesita volver a la versión anterior del paquete, la versión de producción de la aplicación puede descartarse con la sentencia DROP PACKAGE, que hace que se invoque la versión anterior de la aplicación (módulo de carga o ejecutable) que se vinculó utilizando el esquema BACKUP (aquí se pueden utilizar técnicas de vía de acceso de aplicación, específicas de cada plataforma de sistema operativo).

Nota: Este ejemplo asume que la única diferencia entre las versiones del paquete están en las opciones BIND que se utilizaron para crear los paquetes (es decir, no hay diferencias en el código ejecutable).

La aplicación no utiliza la sentencia SET CURRENT PACKAGESET para seleccionar el esquema que desea. En su lugar, permite a DB2 seleccionar el paquete buscándolo en los esquemas listados en el registro especial CURRENT PACKAGE PATH.

Nota: El proceso de precompilación de DB2 Universal Database para OS/390 y z/OS almacena una señal de coherencia en el DBRM (que puede definirse utilizando la opción LEVEL) y durante la resolución de paquetes, se efectúa una comprobación para asegurarse de que la señal de coherencia del

programa corresponde al paquete. Similarmente, el proceso de vinculación de DB2 Universal Database para Linux, UNIX y Windows almacena una indicación horaria en el archivo de vinculación. DB2 Universal Database para Linux, UNIX y Windows también da soporte a una opción LEVEL.

Otro motivo para crear varias versiones de un paquete en esquemas diferentes podría ser el de hacer que entren en vigor diferentes opciones de BIND. Por ejemplo, pueden utilizarse calificadores diferentes para referencias de nombre sin calificar en el paquete.

Las aplicaciones se escriben a menudo con nombres de tabla sin calificar. Esta acción da soporte a múltiples tablas que tienen estructuras y nombres de tabla idénticos, pero calificadores diferentes para distinguir instancias diferentes. Por ejemplo, se pueden haber creado los mismos objetos en un sistema de prueba y en un sistema de producción, pero esos objetos pueden tener calificadores diferentes (por ejemplo, PROD y TEST). Otro ejemplo es una aplicación que particione datos horizontalmente en tablas diferentes entre sistemas DB2 diferentes, cada uno de ellos con un calificador diferente (por ejemplo, EAST, WEST, NORTH, SOUTH; COMPANYA, COMPANYB; Y1999, Y2000, Y2001.). En DB2 Universal Database para OS/390 y z/OS, especifique el calificador de tabla utilizando la opción QUALIFIER del mandato BIND. Cuando se utilice la opción QUALIFIER, los usuarios no tendrán que mantener múltiples programas, cada uno de ellos especificará los nombres totalmente calificados que se necesiten para acceder a las tablas sin calificar. En vez de eso, puede accederse al paquete correcto en tiempo de ejecución emitiendo la sentencia SET CURRENT PACKAGESET desde la aplicación y especificando un único nombre de esquema. Sin embargo, si utiliza SET CURRENT PACKAGESET, seguirá siendo necesario conservar y modificar múltiples aplicaciones: cada una de ellas con su propia sentencia SET CURRENT PACKAGESET para acceder al paquete necesario. Si en vez de eso se emite una sentencia SET CURRENT PACKAGE PATH, se podría listar la totalidad de los esquemas. En el momento de la ejecución, DB2 podría seleccionar el paquete correcto.

Nota: DB2 Universal Database para Linux, UNIX y Windows también da soporte a una opción de vinculación QUALIFIER. Sin embargo, la opción de vinculación QUALIFIER sólo afecta al SQL estático o a los paquetes que utilizan la opción DYNAMICRULES del mandato BIND.

Resolución de nombres de tabla no calificados

Puede manejar nombres de tabla no calificados en la aplicación utilizando uno de los siguientes métodos:

- Para cada usuario, vincule el paquete con distintos parámetros de COLLECTION procedentes de los distintos identificadores de autorización utilizando los siguientes mandatos:

  ```
  CONNECT TO db_name USER user_name
  BIND file_name COLLECTION schema_name
  ```

  En el ejemplo anterior, *db_name* es el nombre de la base de datos, *user_name* es el nombre del usuario y *file_name* es el nombre de la aplicación que se va a vincular. Tenga en cuenta que *user_name* y *schema_name* suelen tener el mismo valor. Luego utilice la sentencia SET CURRENT PACKAGESET para especificar qué paquete utilizar y, por lo tanto, qué calificadores se utilizarán. El calificador por omisión es el identificador de autorización que se utiliza al vincular el paquete.

- Cree vistas para cada usuario con el mismo nombre que la tabla de modo que los nombres de tabla no calificados se resuelvan correctamente.
- Cree un alias para que cada usuario apunte a la tabla deseada.

Información relacionada:

- "Sentencia SET CURRENT PACKAGESET" en la publicación *Consulta de SQL, Volumen 2*
- "Mandato BIND" en la publicación *Consulta de mandatos*

Consideraciones adicionales cuando se vincula

Si la página de códigos de la aplicación utiliza una página de códigos distinta de la de la base de datos, es posible que deba tener en cuenta qué página de códigos utilizar al vincular.

Si la aplicación emite llamadas a alguna de las API de programas de utilidad del gestor de bases de datos, como por ejemplo IMPORT o EXPORT, debe vincular los archivos de vinculación suministrados por el programa de utilidad a la base de datos.

Puede utilizar opciones de vinculación para controlar determinadas operaciones que se producen durante la vinculación, como en los siguientes ejemplos:

- La opción de vinculación QUERYOPT aprovecha una clase de optimización específica al vincular.
- La opción de vinculación EXPLSNAP almacena información de instantánea de Explain para las sentencias de SQL que se pueden elegir en las tablas de Explain.
- La función de vinculación FUNCPATH resuelve correctamente tipos diferenciados definidos por el usuario y funciones definidas por el usuario en SQL estático.

Si el proceso de vinculación comienza pero nunca vuelve, es posible que otras aplicaciones conectadas a la base de datos mantengan bloqueos que necesita. En este caso, asegúrese de que no haya aplicaciones conectadas a la base de datos. Si las hay, desconecte todas las aplicaciones en el servidor y el proceso de vinculación continuará.

Si la aplicación va a acceder a un servidor utilizando DB2 Connect, puede utilizar las opciones de `BIND` disponibles para dicho servidor.

Los archivos de vinculación no son compatibles con versiones anteriores de DB2 Universal Database. En entornos de nivel mixto, DB2® sólo puede utilizar las funciones disponibles al nivel inferior del entorno de base de datos. Por ejemplo, si un cliente V8 se conecta a un servidor V7.2, el cliente sólo podrá utilizar funciones de V7.2. Como los archivos de vinculación expresan las funciones de la base de datos, están sujetos a la restricción de nivel mixto.

Si tiene que volver a vincular archivos de vinculación de nivel superior en sistemas de nivel inferior, puede:

- Utilizar un DB2 Application Development Client de nivel inferior con el servidor de nivel superior y crear archivos de vinculación que se puedan suministrar y vincular al entorno de DB2 Universal Database de nivel inferior.

- Utilizar un cliente DB2 de nivel superior en el entorno de producción de nivel inferior para vincular los archivos de vinculación de nivel superior creados en el entorno de prueba. El cliente de nivel superior sólo pasa las opciones que se aplican al servidor de nivel inferior.

Conceptos relacionados:

- "Binding utilities to the database" en la publicación *Administration Guide: Implementation*
- "Conversión de caracteres entre páginas de códigos diferentes" en la página 650
- "Sustitución de caracteres durante conversiones de páginas de códigos" en la página 651
- "Factor de expansión de conversión de página de códigos" en la página 653

Información relacionada:

- "Mandato BIND" en la publicación *Consulta de mandatos*

Ventajas de la vinculación diferida

La precompilación con la vinculación habilitada permite que una aplicación acceda únicamente a la base de datos utilizada durante el proceso de precompilación. Sin embargo, la precompilación con vinculación diferida permite que una aplicación acceda a muchas bases de datos, puesto que puede vincular el archivo BIND contra cada una. Este método de desarrollo de aplicaciones es más flexible por el hecho de que las aplicaciones sólo se precompilan una vez, pero la aplicación se puede vincular a una base de datos en cualquier momento.

Utilizar la API BIND durante la ejecución permite que una aplicación se vincule a sí misma, quizás como parte de un procedimiento de instalación o antes de que se ejecute un módulo asociado. Por ejemplo, una aplicación puede realizar varias tareas, de las cuales sólo una necesita el uso de sentencias de SQL. Puede diseñar la aplicación de modo que se vincule a sí misma a una base de datos sólo cuando la aplicación llame a la tarea que necesita sentencias de SQL y sólo si aún no existe un paquete asociado.

Otra ventaja del método de vinculación diferida es que le permite crear paquetes sin suministrar el código fuente a los usuarios finales. Puede suministrar los archivos de vinculación asociados con la aplicación.

Información relacionada:

- "sqlabndx - Bind" en la publicación *Administrative API Reference*

Contenido del archivo de vinculación

Con el programa de utilidad de Descripción de archivos de vinculación de DB2® (`db2bfd`), puede visualizar fácilmente el contenido de un archivo de vinculación para examinar y verificar las sentencias de SQL que contiene, así como visualizar las opciones de precompilación utilizadas para crear el archivo de vinculación. Esto puede resultar útil en la determinación de problemas relacionados con el archivo de vinculación de la aplicación.

Información relacionada:

- "db2bfd - Mandato Herramienta de descripción de archivo de vinculación" en la publicación *Consulta de mandatos*

Relaciones entre aplicación, archivo de vinculación y paquete

Un paquete es un objeto almacenado en la base de datos que incluye información necesaria para ejecutar sentencias de SQL específicas en un solo archivo fuente. Una aplicación de base de datos utiliza un paquete para cada archivo fuente precompilado utilizado para crear la aplicación. Cada paquete constituye una entidad separada y no tiene ninguna relación con ningún otro paquete utilizado por la misma o por otras aplicaciones. Los paquetes se crean ejecutando el precompilador contra un archivo fuente con la vinculación habilitada o ejecutando el vinculador posteriormente con uno o más archivos de vinculación.

Las aplicaciones de bases de datos utilizan paquetes por algunas de las mismas razones por las que se compilan las aplicaciones: mejora de rendimiento y de compactación. Al precompilar una sentencia de SQL, la sentencia se compila en el paquete cuando se crea la aplicación en lugar de hacerlo en el tiempo de ejecución. Cada sentencia se analiza y se almacena en el paquete una serie de operandos interpretados de forma más eficiente. Durante la ejecución, el código que ha generado el precompilador llama a las API del gestor de bases de datos de servicios de tiempo de ejecución con la información sobre variables necesaria para la entrada o salida de datos y la información almacenada en el paquete se ejecuta.

Las ventajas de la precompilación sólo se aplican a sentencias de SQL estático. Las sentencias de SQL que se ejecutan de forma dinámica (utilizando PREPARE y EXECUTE o EXECUTE IMMEDIATE) no se precompilan; por lo tanto, deben pasar por todo el conjunto de pasos de proceso en el tiempo de ejecución.

Nota: No dé por supuesto que una versión estática de una sentencia de SQL se ejecuta automáticamente más rápido que la misma sentencia procesada de forma dinámica. En algunos casos, SQL estático es más rápido debido al proceso general necesario para preparar la sentencia dinámica. En otros casos, la misma sentencia preparada de forma dinámica se ejecuta más rápido, porque el optimizador puede utilizar las estadísticas actuales de la base de datos en lugar de las estadísticas de base de datos disponibles en el momento de la vinculación anterior. Observe que si la transacción tarda menos de un par de segundos en completarse, SQL estático suele ser más rápido. Para elegir qué método utilizar, debe crear un prototipo de ambas formas de vinculación.

Conceptos relacionados:

• "SQL dinámico frente a SQL estático" en la página 116

Indicaciones horarias generadas por el precompilador

Al generar un paquete o un archivo de vinculación, el precompilador genera una indicación horaria. La indicación horaria se almacena en el archivo de vinculación o paquete y en el archivo fuente modificado. La indicación horaria también recibe el nombre de *señal de coherencia*.

Cuando una aplicación se precompila con la vinculación habilitada, el paquete y el archivo fuente modificado se generan con indicaciones horarias que coinciden. Si existen varias versiones de un paquete (utilizando la opción PRECOMPILE VERSION), cada versión contendrá una indicación horaria asociada. Cuando se ejecuta una aplicación, el nombre del paquete, el creador y la indicación horaria se envían al gestor de bases de datos, el cual comprueba si hay algún paquete cuyo

nombre, creador e indicación horaria coinciden con los enviados por la aplicación. Si no existe dicha coincidencia, se devuelve uno de los siguientes códigos de error de SQL a la aplicación:

- SQL0818N (conflicto de indicaciones horarias). Este error se devuelve si se encuentra un solo paquete que coincide con el nombre y creador (pero no con la señal de coherencia) y el paquete tiene la versión "" (una serie vacía)
- SQL0805N (paquete no encontrado). Este error se devuelve en las demás situaciones.

Recuerde que cuando vincula una aplicación a una base de datos, los ocho primeros caracteres del nombre de la aplicación se utilizan como el nombre del paquete *a no ser que altere temporalmente el valor por omisión utilizando la opción PACKAGE USING en el mandato PREP*. Asimismo, el ID de versión será "" (una serie vacía), a no ser que se especifique mediante la opción VERSION del mandato PREP. Esto significa que si precompila y vincula dos programas utilizando el mismo nombre sin cambiar el ID de versión, el segundo paquete sustituirá al paquete del primero. Cuando ejecute el primer programa, obtendrá un error de indicación horaria o de paquete no encontrado porque la indicación horaria correspondiente al archivo fuente modificado ya no coincide con la del paquete ene la base de datos. El error de paquete no encontrado puede proceder del uso de la opción de precompilación o de vinculación ACTION REPLACE REPLVER, como en el siguiente ejemplo:

1. Precompile y vincule el paquete SCHEMA1.PKG especificando VERSION VER1. Luego genere la aplicación asociada A1.
2. Precompile y vincule el paquete SCHEMA1.PKG, especificando VERSION VER2 ACTION REPLACE REPLVER VER1. Luego genere la aplicación asociada A2.

   La segunda precompilación y vinculación genera un paquete SCHEMA1.PKG que tiene para VERSION el valor VER2 y la especificación de ACTION REPLACE REPLVER VER1 elimina el paquete SCHEMA1.PKG que tenía para VERSION el valor VER1.

   Si se intenta ejecutar la primera aplicación, se producirá una falta de coincidencia de paquetes y el intento fallará.

Un síntoma parecido sucedería en el siguiente ejemplo:

1. Precompile y vincule el paquete SCHEMA1.PKG, especificando VERSION VER1. Luego genere la aplicación asociada A1.
2. Precompile y vincule el paquete SCHEMA1.PKG, especificando VERSION VER2. Luego genere la aplicación asociada A2.

   En este momento es posible ejecutar ambas aplicaciones, A1 y A2, que se ejecutarían desde los paquetes SCHEMA1.PKG con versiones VER1 y VER2 respectivamente. Si, por ejemplo, se elimina el primer paquete (utilizando la sentencia DROP PACKAGE SCHEMA1.PKG VERSION VER1 SQL), el intento de ejecutar la aplicación A1 fallaría con un error de paquete no encontrado.

Cuando un archivo fuente se precompila pero no se crea un paquete respectivo, se genera un archivo de vinculación y un archivo fuente modificado con indicaciones horarias coincidentes. Para ejecutar la aplicación, el archivo de vinculación se vincula en un paso BIND independiente para crear un paquete y el archivo fuente modificado se compila y se enlaza. Para una aplicación que necesita varios módulos fuente, el proceso de vinculación se debe realizar para cada archivo de vinculación.

En este escenario de vinculación diferida, las indicaciones horarias de la aplicación y del paquete coinciden porque el archivo de vinculación contiene la misma indicación horaria que el que se almacenó en el archivo fuente modificado durante la precompilación.

Conceptos relacionados:

- "Creación de paquetes mediante el mandato BIND" en la página 71

Revinculación de paquetes

Revincular es el proceso de volver a crear un paquete correspondiente a un programa de aplicación que se había vinculado previamente. Debe revincular paquetes si se han marcado como no válidos o no operativos. Sin embargo, en algunas situaciones es posible que desee revincular paquetes que son válidos. Por ejemplo, es posible que desee aprovechar un índice recién creado o utilizar estadísticas actualizadas después de ejecutar el mandato RUNSTATS.

Los paquetes pueden depender de varios tipos de objetos de bases de datos como tablas, vistas, alias, índices, activadores, restricciones referenciales y restricciones de comprobación de tabla. Si un paquete depende de un objeto de base de datos (como una tabla, vista, activador, etc) y dicho objeto se elimina, el paquete se coloca en estado *no válido*. Si el objeto que se elimina es una UDF, el paquete se coloca en estado *no operativo*.

El gestor de bases de datos revincula de forma implícita (o automática) los paquetes no válidos cuando se ejecutan. Los paquetes no operativos se deben revincular de forma explícita ejecutando el mandato `BIND` o el mandato `REBIND`. Tenga que cuenta que la revinculación implícita puede ocasionar errores inesperados si la revinculación implícita falla. Es decir, se devuelve el error de revinculación implícita en la sentencia que se ejecuta, que puede no ser la sentencia que realmente contiene el error. Si se intenta ejecutar un paquete no operativo, se produce un error. Puede decidir revincular de forma explícita paquetes no válidos en lugar de dejar que el sistema los revincule automáticamente. Esto le permite controlar cuándo se produce la revinculación.

La opción de qué mandato utilizar para revincular de forma explícita un paquete depende de las circunstancias. Debe utilizar el mandato `BIND` para revincular un paquete correspondiente a un programa que se ha modificado para incluir más o menos sentencias de SQL o sentencias de SQL modificadas. También debe utilizar el mandato `BIND` si tiene que cambiar opciones de vinculación con respecto a los valores con los que se vinculó originalmente el paquete. En todos los demás casos, utilice el mandato `BIND` o `REBIND`. Debe utilizar `REBIND` cuando la situación no necesite de forma específica el uso de `BIND`, puesto que el rendimiento de `REBIND` es significativamente mejor que el de `BIND`.

Si coexisten varias versiones del mismo nombre de paquete en el catálogo, no se puede revincular más una versión simultáneamente.

Conceptos relacionados:

- "Statement dependencies when changing objects" en la publicación *Administration Guide: Implementation*

Información relacionada:

- "Mandato BIND" en la publicación *Consulta de mandatos*
- "Mandato REBIND" en la publicación *Consulta de mandatos*

Capítulo 4. Cómo escribir programas de SQL estático

Características de SQL estático y razones para utilizarlo 83
Ventajas del SQL estático 84
Programa de SQL estático de ejemplo 85
Recuperación de datos en programas de SQL estático 86
| Efectos de REOPT en el SQL estático 87
Variables del lenguaje principal en programas de SQL estático 87
Variables del lenguaje principal en SQL estático 87
Declaración de variables del lenguaje principal en programas de SQL estático 89
Cómo hacer referencia a variables del lenguaje principal en programas de SQL estático 90
Variables de indicador en programas de SQL estático 91
Inclusión de variables de indicador en programas de SQL estático 91
Tipos de datos para variables de indicador en programas de SQL estático 93
Ejemplo de variable de indicador en un programa de SQL estático 95
Selección de varias filas mediante un cursor . . . 97
Selección de varias filas utilizando un cursor . . 97
Declaración y utilización de cursores en programas de SQL estático 97
Consideraciones sobre tipos de cursor y unidad de trabajo 98
Ejemplo de un cursor en un programa de SQL estático 100
Manipulación de datos recuperados 102
Actualización y supresión de datos recuperados en programas de SQL estático 102
Tipos de cursor 102
Ejemplo de captación en un programa de SQL estático 103
Desplazamiento por datos recuperados y manipulación de los mismos 104
Desplazamiento por datos recuperados previamente 104
Cómo conservar una copia de los datos . . . 105
Recuperación de datos por segunda vez . . . 105
Diferencias en el orden de filas entre la primera y la segunda tabla de resultados 106
Colocación de un cursor al final de una tabla 107
Actualización de datos recuperados previamente 108
Ejemplo de inserción, actualización y supresión en un programa de SQL estático 108
Información de diagnóstico 110
Códigos de retorno 110
Información de error en los campos SQLCODE, SQLSTATE y SQLWARN 110
Truncamiento de símbolos en la estructura SQLCA 111
Consideraciones sobre el manejador de excepciones, señales e interrupciones 111
Consideraciones sobre las rutinas de lista de salida 112
Recuperación de mensajes de error en una aplicación 112

Características de SQL estático y razones para utilizarlo

Cuando la sintaxis de las sentencias de SQL incorporado se conoce por completo en el momento de la precompilación, las sentencias reciben el nombre de SQL *estático*. Esto es para distinguirlas de las sentencias de SQL *dinámico*, cuya sintaxis no se conoce hasta el tiempo de ejecución.

Nota: SQL estático no recibe soporte en lenguajes interpretados, como REXX.

La estructura de una sentencia de SQL se debe especificar por completo para que una sentencia se considere estática. Por ejemplo, los nombres de las columnas y tablas a las que se hace referencia en una sentencia se deben conocer por completo en el momento de la precompilación. La única información que se puede especificar en el tiempo de ejecución son los valores de las variables del lenguaje principal a las que hace referencia la sentencia. Sin embargo, la información sobre variables del lenguaje principal, como tipos de datos, debe estar ya precompilada.

Cuando se prepara una sentencia de SQL estático, se crea un formato ejecutable de la sentencia y se almacena en el paquete en la base de datos. El formato ejecutable se puede construir en el momento de la precompilación o en un momento de vinculación posterior. En cualquier caso, la preparación se produce *antes* del tiempo de ejecución. Se utiliza la autorización de la persona que vincula la aplicación y la

optimización se basa en estadísticas de base de datos y en parámetros de configuración que pueden no estar actualizados cuando se ejecuta la aplicación.

Ventajas del SQL estático

La programación utilizando SQL estático requiere menos esfuerzo que cuando se utiliza SQL dinámico incorporado. Las sentencias de SQL estático simplemente se incorporan en el archivo fuente del lenguaje principal y el precompilador maneja la conversión necesaria a llamadas a API de servicios de tiempo de ejecución del gestor de bases de datos que el compilador del lenguaje principal pueda procesar.

Puesto que se utiliza la autorización de la persona que vincula la aplicación, el usuario final no necesita privilegios directos para ejecutar las sentencias del paquete. Por ejemplo, una aplicación puede permitir que un usuario actualice partes de una tabla sin otorgar un privilegio de actualización sobre la tabla entera. Esto se puede conseguir restringiendo las sentencias de SQL estático para permitir actualizaciones sólo en ciertas columnas o en un rango de valores.

Las sentencias de SQL estático son *permanentes*, lo que significa que las sentencias duran mientras exista el paquete.

Las sentencias de SQL dinámico se colocan en antememoria hasta que dejan de ser válidas, se liberan por razones de gestión de espacio o se cierra la base de datos. Si hace falta, el compilador de SQL de DB2® vuelve a compilar de forma implícita las sentencias de SQL dinámico cuando una sentencia colocada en antememoria deja de ser válida.

La ventaja clave de SQL estático, con respecto a permanencia, es que las sentencias estáticas existen después de que se cierre una determinada base de datos, mientras que las sentencias de SQL dinámico dejan de existir cuando esto sucede. Además, el compilador de SQL de DB2 no tiene que compilar SQL estático en el tiempo de ejecución, mientras que SQL dinámico se tiene que compilar de forma explícita en el tiempo de ejecución (por ejemplo, mediante la sentencia PREPARE). Puesto que DB2 coloca en antememoria sentencias de SQL dinámico, DB2 no tiene que compilar con frecuencia las sentencias, aunque se tienen que compilar al menos una vez cuando el usuario ejecuta la aplicación.

SQL estático puede proporcionar ventajas de rendimiento. Para programas de SQL sencillos de corta ejecución, una sentencia de SQL estático se ejecuta más rápido que la misma sentencia procesada de forma dinámica porque el proceso general de preparación de un formato ejecutable de la sentencia se realiza en el momento de la precompilación en lugar de en el tiempo de ejecución.

Nota: El rendimiento de SQL estático depende de las estadísticas de la base de datos la última vez que se vinculó la aplicación. Sin embargo, si estas estadísticas cambian, el rendimiento de SQL dinámico equivalente puede ser muy diferente. Si, por ejemplo, se añade un índice a una base de datos posteriormente, una aplicación que utilice SQL estático no puede aprovechar el índice a no ser que se revincule a la base de datos. Además, si utiliza variables del lenguaje principal en una sentencia de SQL estático, el optimizador no podrá aprovechar ninguna de las estadísticas de distribución correspondientes a la tabla.

Información relacionada:

- "Sentencia EXECUTE" en la publicación *Consulta de SQL, Volumen 2*

Programa de SQL estático de ejemplo

Este programa de ejemplo muestra sentencias de SQL estático y llamadas a API del gestor de bases de datos en los lenguajes C/C++, Java™ y COBOL.

El ejemplo en C/C++ y Java consulta la tabla **org** de la base de datos SAMPLE para buscar el nombre y número del departamento situado en Nueva York y luego coloca esos datos en variables de lenguaje principal.

El ejemplo en COBOL consulta la tabla **employee** de la base de datos sample para buscar el nombre del empleado cuyo apellido es Johnson y luego coloca el nombre en una variable del lenguaje principal.

Nota: El lenguaje REXX no da soporte a SQL estático, así que no se proporciona ningún ejemplo.

- C/C++ (**tbread**)

```
SELECT deptnumb, deptname INTO :deptnumb, :deptname
FROM org
WHERE location = 'New York'
```

  Esta consulta está en la función `TbRowSubselect()` del ejemplo. Para obtener más información, consulte los siguientes ejemplos relacionados.

- Java (TbRead.sqlj)

```
#sql cur2 = {SELECT deptnumb, deptname
 FROM org
 WHERE location = 'New York'};
 // captar el cursor
 #sql {FETCH :cur2 INTO :deptnumb, :deptname};
```

  Esta consulta está en la función `rowSubselect()` del ejemplo **TbRead.sqlj**. Para obtener más información, consulte los siguientes ejemplos relacionados.

- COBOL (**static.sqb**)

  El ejemplo **static** contiene ejemplos de sentencias de SQL estático y llamadas a API del gestor de bases de datos en el lenguaje COBOL. La sentencia SELECT INTO selecciona una fila de datos de tablas de una base de datos, y los valores de esta fila se asignan a variables del lenguaje principal especificadas en la sentencia. Por ejemplo, la siguiente sentencia distribuye el nombre del empleado cuyo apellido es JOHNSON a la variable del lenguaje principal `firstname`:

```
SELECT FIRSTNME
 INTO :firstname
 FROM EMPLOYEE
 WHERE LASTNAME = 'JOHNSON'
```

Conceptos relacionados:

- "Recuperación de datos en programas de SQL estático" en la página 86
- "Recuperación de mensajes de error en una aplicación" en la página 112

Tareas relacionadas:

- "Declaración de variables del lenguaje principal en programas de SQL estático" en la página 89
- "Selección de varias filas utilizando un cursor" en la página 97
- "Configuración de la base de datos de ejemplo" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Información relacionada:

- "Sentencia SELECT INTO" en la publicación *Consulta de SQL, Volumen 2*

Ejemplos relacionados:

- "dtlob.out -- HOW TO USE THE LOB DATA TYPE (C)"
- "tbinfo.out -- HOW TO GET INFORMATION AT THE TABLE LEVEL (C)"
- "tbread.out -- HOW TO READ TABLES (C)"
- "tbread.sqc -- How to read tables (C)"
- "dtlob.sqC -- How to use the LOB data type (C++)"
- "tbinfo.sqC -- How to get information at the table level (C++)"
- "tbread.out -- HOW TO READ TABLES (C++)"
- "tbread.sqC -- How to read tables (C++)"
- "static.sqb -- Get table data using static SQL statement (IBM COBOL)"
- "static.sqb -- Get table data using static SQL statement (MF COBOL)"
- "TbRead.out -- HOW TO READ TABLE DATA. Connect to 'sample' database using JDBC type 2 driver (JDBC)"
- "TbRead.sqlj -- How to read table data (SQLj)"

Recuperación de datos en programas de SQL estático

Una de las tareas más comunes de un programa de aplicación SQL es recuperar datos. Esta tarea se lleva a cabo utilizando la *sentencia select*, que es una forma de consulta que busca filas de tablas de la base de datos que cumplen con las condiciones de búsqueda especificadas. Si existen dichas filas, los datos se recuperan y se colocan en variables especificadas en el programa de sistema principal, donde se pueden utilizar con el fin para el que está diseñado.

Después de escribir una sentencia select, codifique las sentencias de SQL que definen el modo en que la información se pasará a la aplicación.

Puede entender el resultado de una sentencia select como una tabla que tiene filas y columnas, parecida a una tabla de la base de datos. Si sólo se devuelve una fila, puede distribuir los resultados directamente en variables del lenguaje principal especificadas por la sentencia SELECT INTO.

Si se devuelve más de una fila, debe utilizar un *cursor* para captarlas de una en una. Un cursor es una estructura de control con nombre que es utilizada por un programa de aplicación para apuntar a una fila específica dentro de un conjunto ordenado de filas.

Conceptos relacionados:

- "Variables del lenguaje principal en SQL estático" en la página 87
- "Ejemplo de un cursor en un programa de SQL estático" en la página 100

Tareas relacionadas:

- "Declaración de variables del lenguaje principal en programas de SQL estático" en la página 89
- "Cómo hacer referencia a variables del lenguaje principal en programas de SQL estático" en la página 90
- "Inclusión de variables de indicador en programas de SQL estático" en la página 91

- "Selección de varias filas utilizando un cursor" en la página 97
- "Declaración y utilización de cursores en programas de SQL estático" en la página 97

Efectos de REOPT en el SQL estático

La opción de vinculación REOPT puede hacer que sentencias de SQL estático que contengan variables de lenguaje principal o registros especiales se comporten como sentencias de vinculación incremental. Esto significa que estas sentencias se compilan cuando se ejecuta EXECUTE u OPEN, en lugar de hacerlo durante la vinculación. Durante esta compilación, se selecciona el plan de acceso, de acuerdo con los valores reales de estas variables.

Cuando se utiliza REOPT ONCE, el plan de acceso se coloca en antememoria después de la primera petición OPEN o EXECUTE, y se utiliza para la ejecución subsiguiente de esta sentencia. Cuando se utiliza REOPT ALWAYS, el plan de acceso se regenera para cada petición OPEN y EXECUTE, y para crear el plan se utiliza el conjunto actual de valores de variables de lenguaje principal, marcadores de parámetros y registros especiales.

Variables del lenguaje principal en programas de SQL estático

Las secciones siguientes describen cómo utilizar variables del lenguaje principal en programas de SQL estático.

Variables del lenguaje principal en SQL estático

Las *variables del lenguaje principal* son variables a las que hacen referencia las sentencias de SQL incorporado. Transmiten datos entre el gestor de bases de datos y un programa de aplicación. Cuando utilice una variable lenguaje principal en una sentencia de SQL, debe preceder su nombre con un signo de dos puntos (**:**). Cuando utilice una variable del lenguaje principal en una sentencia de lenguaje principal, omita el signo de dos puntos.

Las variables del lenguaje principal se declaran en lenguajes principales compilados y están delimitadas por las sentencias BEGIN DECLARE SECTION y END DECLARE SECTION. Estas sentencias permiten al precompilador encontrar las declaraciones.

Nota: Los programas Java™ JDBC y SQLJ no utilizan secciones declare. Las variables del lenguaje principal en Java siguen la sintaxis normal de declaración de variables Java.

Las variables del lenguaje principal se declaran utilizando un subconjunto del lenguaje principal.

Las siguientes normas se aplican a secciones de declaración de variables del lenguaje principal:

- Todas las variables del lenguaje principal se deben declarar en el archivo fuente antes de que se haga referencia a las mismas, excepto para las variables del lenguaje principal que hacen referencia a estructuras SQLDA.
- Se pueden utilizar varias secciones declare en un archivo fuente.
- El precompilador desconoce las normas de establecimiento de ámbito de variables del lenguaje principal.

Con respecto a sentencias de SQL, todas las variables del lenguaje principal tienen ámbito global independientemente de dónde se declaren realmente en un solo archivo fuente. Por lo tanto, los nombres de variables del lenguaje principal deben ser exclusivos dentro de un archivo fuente.

Esto no significa que el precompilador de DB2® cambie el ámbito de variables del lenguaje principal por global para que se pueda acceder a las mismas fuera del ámbito en el que se han definido. Supongamos que tenemos el siguiente ejemplo:

```
foo1(){
  .
  .
  .
  BEGIN SQL DECLARE SECTION;
  int x;
  END SQL DECLARE SECTION;
x=10;
  .
  .
  .
}


foo2(){
  .
  .
  .
  y=x;
  .
  .
  .
}
```

En función del lenguaje, el ejemplo anterior no se podrá compilar porque la variable `x` no está declarada en la función `foo2()` o el valor de `x` no se establecerá en 10 en `foo2()`. Para evitar este problema, debe declarar `x` como una variable global o pasar `x` como parámetro a la función `foo2()` del siguiente modo:

```
  foo1(){
  .
  .
  .
    BEGIN SQL DECLARE SECTION;
    int x;
    END SQL DECLARE SECTION;
    x=10;
    foo2(x);
  .
  .
  .
  }


  foo2(int x){
  .
  .
  .
    y=x;
  .
  .
  .
  }
```

Conceptos relacionados:

- "Variables del lenguaje principal en C y C++" en la página 152
- "Variables del lenguaje principal en COBOL" en la página 198
- "Variables del lenguaje principal en FORTRAN" en la página 221
- "Variables del lenguaje principal en REXX" en la página 533

Tareas relacionadas:

- "Declaración de variables del lenguaje principal con el Generador de declaraciones db2dclgn" en la página 32
- "Declaración de variables del lenguaje principal en programas de SQL estático" en la página 89
- "Cómo hacer referencia a variables del lenguaje principal en programas de SQL estático" en la página 90

Declaración de variables del lenguaje principal en programas de SQL estático

Declare variables de lenguaje principal del programa para que se puedan utilizar para transferir datos entre el gestor de bases de datos y la aplicación.

Procedimiento:

Declare las variables del lenguaje principal utilizando la sintaxis correspondiente al lenguaje principal que esté utilizando. La tabla siguiente contiene ejemplos.

Tabla 4. Declaraciones de variables del lenguaje principal por lenguaje principal

Lenguaje	Código fuente de ejemplo

C/C++	`EXEC SQL BEGIN DECLARE SECTION;`
	`  short  dept=38, age=26;`
	`  double  salary;`
	`  char  CH;`
	`  char  name1[9], NAME2[9];`
	`  /* Comentario de C */`
	`  short  nul_ind;`
	`EXEC SQL END DECLARE SECTION;`

Java	`// Observe que las declaraciones de variables de lenguaje`
	`// principal de Java siguen las reglas normales de declaración`
	`// de variables Java y no tienen ningún equivalente de`
	`// DECLARE SECTION`
	`  short  dept=38, age=26;`
	`  double  salary;`
	`  char  CH;`
	`  String  name1[9], NAME2[9];`
	`  /* Comentario de Java */`
	`  short  nul_ind;`

COBOL	`EXEC SQL BEGIN DECLARE SECTION END-EXEC.`
	`  01 age  PIC S9(4) COMP-5 VALUE 26.`
	`  01 DEPT  PIC S9(9) COMP-5 VALUE 38.`
	`  01 salary  PIC S9(6)V9(3) COMP-3.`
	`  01 CH  PIC X(1).`
	`  01 name1  PIC X(8).`
	`  01 NAME2  PIC X(8).`
	`*  Comentario de COBOL`
	`  01 nul-ind  PIC S9(4) COMP-5.`
	`EXEC SQL END DECLARE SECTION END-EXEC.`

Tabla 4. Declaraciones de variables del lenguaje principal por lenguaje principal (continuación)

Lenguaje	Código fuente de ejemplo

FORTRAN	`EXEC SQL BEGIN DECLARE SECTION`
	`  integer*2  age  /26/`
	`  integer*4  dept  /38/`
	`  real*8  salary`
	`  character  ch`
	`  character*8  name1,NAME2`
	`C  Comentario de FORTRAN`
	`  integer*2  nul_ind`
	`EXEC SQL END DECLARE SECTION`

Tareas relacionadas:

- "Declaración de variables del lenguaje principal con el Generador de declaraciones db2dclgn" en la página 32
- "Cómo hacer referencia a variables del lenguaje principal en programas de SQL estático" en la página 90

Cómo hacer referencia a variables del lenguaje principal en programas de SQL estático

Después de declarar la variable del lenguaje principal, puede hacer referencia a la misma en el programa de aplicación. Cuando utilice una variable del lenguaje principal en una sentencia de SQL, preceda su nombre con un signo de dos puntos (:). Si utiliza una variable del lenguaje principal en una sentencia del lenguaje principal, omita los dos puntos.

Procedimiento:

Haga referencia a las variables del lenguaje principal utilizando la sintaxis correspondiente al lenguaje principal que esté utilizando. La tabla siguiente contiene ejemplos.

Tabla 5. Referencias a variables del lenguaje principal por lenguaje principal

Lenguaje	Código fuente de ejemplo
C/C++	`EXEC SQL FETCH C1 INTO :cm;` `printf( "Commission = %f\n", cm );`
JAVA (SQLJ)	`#SQL { FETCH :c1 INTO :cm };` `System.out.println("Commission = " + cm);`
COBOL	`EXEC SQL FETCH C1 INTO :cm END-EXEC` `DISPLAY 'Commission = ' cm`
FORTRAN	`EXEC SQL FETCH C1 INTO :cm` `WRITE(*,*) 'Commission = ', cm`

Tareas relacionadas:

- "Declaración de variables del lenguaje principal con el Generador de declaraciones db2dclgn" en la página 32
- "Declaración de variables del lenguaje principal en programas de SQL estático" en la página 89

Variables de indicador en programas de SQL estático

Las secciones siguientes describen cómo utilizar las variables de indicador en programas de SQL estático.

Inclusión de variables de indicador en programas de SQL estático

Las aplicaciones escritas en lenguajes que no sean Java deben prepararse para recibir valores nulos asociando una *variable de indicador* con cualquier variable del lenguaje principal que pueda recibir un nulo. Las aplicaciones Java comparan el valor de la variable del lenguaje principal con `null` de Java para determinar si el valor recibido es nulo. El gestor de bases de datos y la aplicación de sistema principal comparten una variable de indicador; por lo tanto, la variable de indicador se debe declarar en la aplicación como una variable del lenguaje principal. Esta variable del lenguaje principal corresponde al tipo SMALLINT de datos de SQL.

Una variable de indicador se coloca en una sentencia de SQL inmediatamente después de la variable del lenguaje principal y va precedida por dos puntos. Un espacio puede separar la variable de indicador de la variable del lenguaje principal, aunque no es necesario. Sin embargo, no coloque una coma entre la variable del lenguaje principal y la variable de indicador. También puede especificar una variable de indicador utilizando la palabra clave INDICATOR opcional, que debe colocar entre la variable del lenguaje principal y su indicador.

Procedimiento:

Utilice la palabra clave INDICATOR para escribir variables de indicador. La tabla siguiente contiene ejemplos correspondientes a los lenguajes principales soportados:

Tabla 6. Variables de indicador por lenguaje principal

Lenguaje	Código fuente de ejemplo

C/C++	`EXEC SQL FETCH C1 INTO :cm INDICATOR :cmind;`
	`if ( cmind < 0 )`
	`  printf( "Commission is NULL\n" );`

JAVA (SQLJ)	`#SQL { FETCH :c1 INTO :cm };`
	`if ( cm == null )`
	`  System.out.println( "Commission is NULL\n" );`

COBOL	`EXEC SQL FETCH C1 INTO :cm INDICATOR :cmind END-EXEC`
	`IF cmind LESS THAN 0`
	`  DISPLAY 'Commission is NULL'`

FORTRAN	`EXEC SQL FETCH C1 INTO :cm INDICATOR :cmind`
	`IF ( cmind .LT. 0 ) THEN`
	`  WRITE(*,*) 'Commission is NULL'`
	`ENDIF`

En los ejemplos anteriores, *cmind* se examina para ver si tiene un valor negativo. Si el valor no es negativo, la aplicación puede utilizar el valor devuelto de *cm*. Si el valor es negativo, el valor captado es NULL y no se debe utilizar *cm*. En este caso, el gestor de bases de datos no cambia el valor de la variable del lenguaje principal.

Nota: Si el parámetro de configuración de base de datos *dft_sqlmathwarn* tiene el valor 'YES', el valor de *cmind* puede ser -2. Este valor indica un valor NULL

causado por la evaluación de una expresión con un error aritmético o por un desbordamiento al intentar convertir el valor de resultado numérico en la variable del lenguaje principal.

Si el tipo de datos puede manejar valores NULL, la aplicación debe proporcionar un indicador de NULL. De lo contrario, puede producirse un error. Si no se utiliza un indicador de valor NULL, se devuelve un SQLCODE -305 (SQLSTATE 22002).

Si la estructura SQLCA indica un aviso de truncamiento, se pueden examinar las variables de indicador para ver si hay algún truncamiento. Si una variable de indicador tiene un valor positivo, significa que se ha producido un truncamiento.

- Si la parte en segundos de un tipo de datos TIME está truncada, el valor del indicador contiene la parte en segundos de los datos truncados.
- Para todos los demás tipos de datos de serie, excepto para objetos grandes (LOB), el valor de indicador representa la longitud real de los datos devueltos. Los tipos diferenciados definidos por el usuario (UDT) se manejan del mismo modo que su tipo base.

Cuando se procesan sentencias INSERT o UPDATE, el gestor de bases de datos comprueba la variable de indicador, si la hay. Si la variable de indicador es negativa, el gestor de bases de datos establece el valor de la columna de destino en NULL, si se permiten valores NULL.

Si la variable de indicador es cero o positiva, el gestor de bases de datos utiliza el valor de la variable del lenguaje principal asociada.

El campo SQLWARN1 de la estructura SQLCA puede contener un valor `X` o `W` si el valor de una columna de serie está truncado cuando se asigna a una variable del lenguaje principal. El campo contiene un valor `N` si un terminador de valores nulos está truncado.

El gestor de bases de datos devuelve un valor `X` sólo si se cumplen todas las condiciones siguientes:

- Existe una conexión de página de códigos combinada en la que la conversión de datos de serie de caracteres de la página de códigos de la base de datos a la página de códigos de la aplicación implica un cambio en la longitud de los datos.
- Un cursor está bloqueado.
- La aplicación proporciona una variable de indicador.

El valor devuelto en la variable de indicador tendrá la longitud de la serie de caracteres resultante en la página de códigos de la aplicación.

En los demás casos en los que interviene un truncamiento de datos (en contraste con el truncamiento del terminador de valores NULL), el gestor de bases de datos devuelve un valor `W`. En este caso, el gestor de bases de datos devuelve un valor en la variable de indicador a la aplicación que tiene la longitud de la serie de caracteres resultante en la página de códigos del elemento de lista select (la página de códigos de la aplicación, la página de códigos de la base de datos o ninguna).

Tareas relacionadas:

- “Declaración de variables del lenguaje principal con el Generador de declaraciones db2dclgn” en la página 32

- "Declaración de variables del lenguaje principal en programas de SQL estático" en la página 89
- "Cómo hacer referencia a variables del lenguaje principal en programas de SQL estático" en la página 90

Información relacionada:

- "Tipos de datos para variables de indicador en programas de SQL estático" en la página 93

Tipos de datos para variables de indicador en programas de SQL estático

A cada columna de cada tabla de DB2 se asigna un *tipo de datos de SQL* cuando se crea la columna. Para obtener información sobre cómo se asignan estos tipos a columnas, consulte la sentencia CREATE TABLE. El gestor de bases de datos da soporte a los siguientes tipos de datos de columna:

SMALLINT
Entero con signo de 16 bits.

INTEGER
Entero con signo de 32 bits. **INT** se puede utilizar como sinónimo de este tipo.

BIGINT
Entero con signo de 64 bits.

DOUBLE
Punto flotante de doble precisión. **DOUBLE PRECISION** y **FLOAT(*n*)** (donde *n* es mayor que 24) son sinónimos de este tipo.

REAL Punto flotante de precisión sencilla. **FLOAT(*n*)** (donde *n* es menor que 24) es un sinónimo de este tipo.

DECIMAL
Decimal empaquetado. **DEC**, **NUMERIC** y **NUM** son sinónimos de este tipo.

CHAR
Serie de caracteres de longitud fija comprendida entre 1 byte y 254 bytes. **CHARACTER** se puede utilizar como sinónimo de este tipo.

VARCHAR
Serie de caracteres de longitud variable comprendida entre 1 byte y 32672 bytes. **CHARACTER VARYING** y **CHAR VARYING** son sinónimos de este tipo.

LONG VARCHAR
Serie de caracteres de longitud variable larga comprendida entre 1 byte y 32700 bytes.

CLOB Serie de caracteres de longitud variable de objetos grandes de longitud comprendida entre 1 byte y 2 gigabytes.

BLOB Serie binaria de longitud variable de objetos grandes de longitud comprendida entre 1 byte y 2 gigabytes.

DATE Serie de caracteres de longitud 10 que representa una fecha.

TIME Serie de caracteres de longitud 8 que representa una hora.

TIMESTAMP
Serie de caracteres de longitud 26 que representa una indicación horaria.

Los siguientes tipos de datos sólo reciben soporte en entornos de juego de caracteres de doble byte (DBCS) y de juego de caracteres Extended UNIX Code (EUC):

GRAPHIC
Serie gráfica de longitud fija comprendida entre 1 y 127 caracteres de doble byte.

VARGRAPHIC
Serie gráfica de longitud variable comprendida entre 1 y 16.336 caracteres de doble byte.

LONG VARGRAPHIC
Serie gráfica de longitud variable larga comprendida entre 1 y 16.350 caracteres de doble byte.

DBCLOB
Serie gráfica de longitud variable de objetos grandes de longitud comprendida entre 1 y 1.073.741.823 caracteres de doble byte.

Notas:

1. Cada tipo de datos soportado puede tener el atributo NOT NULL. Esto se trata como otro tipo.
2. El conjunto anterior de tipos de datos se puede ampliar definiendo tipos diferenciados definidos por el usuario (UDT). Los UDT son tipos de datos separados que utilizan la representación de uno de los tipos de SQL integrados.

Los lenguajes principales soportados tienen tipos de datos que corresponden con la mayoría de los tipos de datos del gestor de bases de datos. Sólo estos tipos de datos de lenguajes principales se pueden utilizar en declaraciones de variables del lenguaje principal. Cuando el precompilador encuentra una declaración de variable del lenguaje principal, determina el valor de tipo de datos de SQL adecuado. El gestor de bases de datos utiliza este valor para convertir los datos intercambiados entre él mismo y la aplicación.

Como programador de aplicaciones, es importante que comprenda el modo en que el gestor de bases de datos maneja comparaciones y asignaciones entre distintos tipos de datos. Explicado de forma sencilla, los tipos de datos deben ser compatibles entre sí durante las operaciones de asignación y comparación, tanto si el gestor de bases de datos trabaja con dos tipos de datos de columnas de SQL como si lo hace con dos tipos de datos de lenguaje principal o con uno de cada.

La norma *general* de la compatibilidad de tipos de datos establece que todos los tipos de datos numéricos soportados del lenguaje de sistema principal se pueden comparar y asignar con todos los tipos de datos numéricos del gestor de bases de datos y que todos los tipos de caracteres del lenguaje de sistema principal son compatibles con todos los tipos de caracteres del gestor de bases de datos; los tipos numéricos son incompatibles con los tipos de caracteres. Sin embargo, también hay algunas excepciones a esta normal general, en función de la idiosincrasia del lenguaje de sistema principal y de las limitaciones impuestas cuando se trabaja con objetos grandes.

Dentro de sentencias de SQL, DB2 proporciona conversiones entre tipos de datos compatibles. Por ejemplo, en la siguiente sentencia SELECT, SALARY y BONUS

son columnas de tipo DECIMAL; sin embargo, la compensación total de cada empleado se devuelve como datos de tipo DOUBLE:

```
SELECT EMPNO, DOUBLE(SALARY+BONUS) FROM EMPLOYEE
```

Observe que la ejecución de la sentencia anterior incluye la conversión entre los tipos de datos DECIMAL y DOUBLE.

Para que los resultados de la consulta resulten más fáciles de leer en pantalla, puede utilizar la siguiente sentencia SELECT:

```
SELECT EMPNO, DIGIT(SALARY+BONUS) FROM EMPLOYEE
```

Para convertir datos dentro de la aplicación, póngase en contacto con el proveedor del compilador para obtener rutinas, clases, tipos integrados o API adicionales que den soporte a esta conversión.

Si la página de códigos de la aplicación no coincide con la página de códigos de la base de datos, es posible que los tipos de datos de caracteres también estén sujetos a la conversión de caracteres.

Conceptos relacionados:

- "Consideraciones sobre la conversión de datos" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*
- "Conversión de caracteres entre páginas de códigos diferentes" en la página 650

Información relacionada:

- "Sentencia CREATE TABLE" en la publicación *Consulta de SQL, Volumen 2*
- "Tipos de datos SQL soportados en C y C++" en la página 180
- "Tipos de datos de SQL soportados en COBOL" en la página 210
- "Tipos de datos SQL soportados en FORTRAN" en la página 227
- "Tipos de datos SQL soportados en REXX" en la página 539
- "Tipos de datos de Java, JDBC y SQL" en la página 395

Ejemplo de variable de indicador en un programa de SQL estático

A continuación se muestran ejemplos de cómo utilizar programas C/C++ de variables de indicador que utilizan SQL estático:

- Ejemplo 1

  El siguiente ejemplo muestra la implantación de variables de indicador en columnas de datos que pueden tener valores null. En este ejemplo, la columna FIRSTNAME no puede contener valores null, pero la columna WORKDEPT sí puede.

```
EXEC SQL BEGIN DECLARE SECTION;
  char wd[3];
  short wd_ind;
  char firstname[13];
EXEC SQL END DECLARE SECTION;

  /* conectar con base de datos de ejemplo */

EXEC SQL SELECT FIRSTNME, WORKDEPT
  INTO :firstname, :wd:wdind
  FROM EMPLOYEE
  WHERE LASTNAME = 'JOHNSON';
```

Puesto que la columna WORKDEPT puede tener un valor null, se debe declarar una variable de indicador como una variable del lenguaje principal antes de que se utilice.

- Ejemplo 2 (**dtlob**)

  El ejemplo **dtlob** tiene una función denominada `BlobFileUse()`. La función `BlobFileUse()` contiene una consulta que lee datos BLOB en un archivo mediante una sentencia SELECT INTO:

```
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS BLOB_FILE blobFilePhoto;
  char photoFormat[10];
  char empno[7];
  short lobind;
EXEC SQL END DECLARE SECTION;

  /* Conectar con la base de datos de ejemplo */

SELECT picture INTO :blobFilePhoto:lobind
  FROM emp_photo
  WHERE photo_format = :photoFormat AND empno = '000130'
```

  Puesto que la columna BLOBFILEPHOTO puede tener un valor null, se debe declarar una variable de indicador LOBIND como una variable del lenguaje principal antes de que se utilice. El ejemplo **dtlob** muestra cómo trabajar con LOB. Consulte los ejemplos para obtener más información sobre cómo utilizar LOB.

Conceptos relacionados:

- "Programa de SQL estático de ejemplo" en la página 85

Tareas relacionadas:

- "Inclusión de variables de indicador en programas de SQL estático" en la página 91

Información relacionada:

- "Tipos de datos para variables de indicador en programas de SQL estático" en la página 93

Ejemplos relacionados:

- "dtlob.out -- HOW TO USE THE LOB DATA TYPE (C)"
- "dtlob.sqc -- How to use the LOB data type (C)"
- "dtlob.out -- HOW TO USE THE LOB DATA TYPE (C++)"
- "dtlob.sqC -- How to use the LOB data type (C++)"

Selección de varias filas mediante un cursor

Las secciones siguientes describen cómo seleccionar filas mediante un cursor. Los programas de ejemplo que muestran cómo declarar un cursor, abrir el cursor, captar filas de la tabla y cerrar el cursor también se describen brevemente.

Selección de varias filas utilizando un cursor

Para permitir que una aplicación recupere un conjunto de filas, SQL utiliza un mecanismo denominado un *cursor*.

Para ayudarle a comprender el concepto de un cursor, supongamos que el gestor de bases de datos crea una *tabla de resultados* que contenga todas las filas recuperadas al ejecutar una sentencia SELECT. Un cursor permite que las filas procedentes de la tabla de resultados estén disponibles para una aplicación, identificando o haciendo referencia a una *fila actual* de esta tabla. Cuando se utiliza un cursor, una aplicación puede recuperar cada fila secuencialmente de la tabla de resultados hasta que se encuentra una condición de fin de datos, es decir, se alcanza la condición NOT FOUND, SQLCODE +100 (SQLSTATE 02000). El conjunto de filas obtenido como resultado de ejecutar la sentencia SELECT puede constar de cero, una o más filas, en función del número de filas que cumplan con la condición de búsqueda.

Procedimiento:

Los pasos a seguir para procesar un cursor son los siguientes:

1. Especifique el cursor utilizando una sentencia DECLARE CURSOR.
2. Realice la consulta y cree la tabla de resultados utilizando la sentencia OPEN.
3. Recupere filas de una en una utilizando la sentencia FETCH.
4. Procese las filas con las sentencias DELETE o UPDATE (si hace falta).
5. Termine el cursor utilizando la sentencia CLOSE.

Una aplicación puede utilizar varios cursores simultáneamente. Cada cursor necesita su propio conjunto de sentencias DECLARE CURSOR, OPEN, CLOSE y FETCH.

Conceptos relacionados:

- "Ejemplo de un cursor en un programa de SQL estático" en la página 100

Declaración y utilización de cursores en programas de SQL estático

Utilice la sentencia DECLARE CURSOR para definir y nombrar el cursor y para identificar el conjunto de filas que se tienen que recuperar mediante una sentencia SELECT.

La aplicación asigna un nombre para el cursor. Se hace referencia a este nombre en siguientes sentencias OPEN, FETCH y CLOSE. La consulta es cualquier sentencia select válida.

Restricciones:

La ubicación de la sentencia DECLARE es arbitraria, pero debe estar antes de la primera vez que se utiliza el cursor.

Procedimiento:

Utilice la sentencia DECLARE para definir el cursor. La tabla siguiente contiene ejemplos correspondientes a los lenguajes principales soportados:

Tabla 7. Declaraciones de cursor por lenguaje principal

Lenguaje	Código fuente de ejemplo

C/C++	`EXEC SQL DECLARE C1 CURSOR FOR`
	`  SELECT PNAME, DEPT FROM STAFF`
	`  WHERE JOB=:host_var;`

JAVA (SQLJ)	`#sql iterator cursor1(host_var data type);`
	`#sql cursor1 = { SELECT PNAME, DEPT FROM STAFF`
	`  WHERE JOB=:host_var };`

COBOL	`EXEC SQL DECLARE C1 CURSOR FOR`
	`  SELECT NAME, DEPT FROM STAFF`
	`  WHERE JOB=:host-var END-EXEC.`

FORTRAN	`EXEC SQL DECLARE C1 CURSOR FOR`
	`+  SELECT NAME, DEPT FROM STAFF`
	`+  WHERE JOB=:host_var`

Conceptos relacionados:

- "Consideraciones sobre tipos de cursor y unidad de trabajo" en la página 98

Tareas relacionadas:

- "Selección de varias filas utilizando un cursor" en la página 97

Información relacionada:

- "Tipos de cursor" en la página 102

Consideraciones sobre tipos de cursor y unidad de trabajo

Las acciones de una operación COMMIT o ROLLBACK varían para los cursores en función del modo en que éstos se declaren:

Cursores de sólo lectura

Si se determina que un cursor es de sólo lectura y utiliza un nivel de aislamiento de lectura repetitiva, se siguen obteniendo y manteniendo bloqueos de lectura repetitiva en las tablas del sistema que necesita la unidad de trabajo. Por lo tanto, es importante que las aplicaciones emitan periódicamente sentencias COMMIT, incluso para cursores de sólo lectura.

Opción WITH HOLD

Si una aplicación completa una unidad de trabajo emitiendo una sentencia COMMIT, el gestor de bases de datos cierra *todos los cursores abiertos*, excepto aquellos declarados utilizando la opción WITH HOLD.

Un cursor declarado con la opción WITH HOLD mantiene los recursos a los que accede en varias unidades de trabajo. El efecto exacto de declarar un cursor WITH HOLD depende del modo en que finaliza la unidad de trabajo:

- Si la unidad de trabajo finaliza con una sentencia COMMIT, los cursores definidos con WITH HOLD permanecen abiertos (OPEN). El cursor se

coloca antes de la siguiente fila lógica de la tabla de resultados. Además, las sentencias preparadas que hacen referencia a cursores OPEN definidos con WITH HOLD se retienen. Sólo las peticiones FETCH y CLOSE asociadas con un determinado cursor son válidas inmediatamente después de la sentencia COMMIT. Las sentencias UPDATE WHERE CURRENT OF y DELETE WHERE CURRENT OF sólo son válidas para filas captadas dentro de la misma unidad de trabajo.

Nota: Si un paquete se revincula durante una unidad de trabajo, todos los cursores retenidos se cierran.

- Si la unidad de trabajo finaliza con una sentencia ROLLBACK, todos los cursores abiertos se cierran, todos los bloqueos adquiridos durante la unidad de trabajo se liberan y todas las sentencias preparadas que dependen del trabajo realizado en dicha unidad se eliminan.

Por ejemplo, supongamos que la tabla TEMPL contiene 1.000 entradas. Desea actualizar la columna salary correspondiente a todos los empleados, y espera emitir una sentencia COMMIT cada vez que actualiza 100 filas.

1. Declare el cursor utilizando la opción WITH HOLD:

```
EXEC SQL DECLARE EMPLUPDT CURSOR WITH HOLD FOR
  SELECT EMPNO, LASTNAME, PHONENO, JOBCODE, SALARY
  FROM TEMPL FOR UPDATE OF SALARY
```

2. Abra el cursor y recupere datos de la tabla de resultados, fila a fila:

```
EXEC SQL OPEN EMPLUPDT
  .
  .
  .

EXEC SQL FETCH EMPLUPDT
  INTO :upd_emp, :upd_lname, :upd_tele, :upd_jobcd, :upd_wage,
```

3. Cuando desee actualizar o suprimir una fila, utilice una sentencia UPDATE o DELETE utilizando la opción WHERE CURRENT OF. Por ejemplo, para actualizar la fila actual, el programa puede emitir:

```
EXEC SQL UPDATE TEMPL SET SALARY = :newsalary
  WHERE CURRENT OF EMPLUPDT
```

4. Después de emitir una sentencia COMMIT, debe emitir una sentencia FETCH antes de poder actualizar otra fila.

Debe incluir código en la aplicación para detectar y manejar un SQLCODE -501 (SQLSTATE 24501), que se puede devolver en una sentencia FETCH o CLOSE si la aplicación:

- Utiliza cursores declarados WITH HOLD.
- Ejecuta más de una unidad de trabajo y deja un cursor WITH HOLD abierto en los límites de la unidad de trabajo (COMMIT WORK).

Si una aplicación invalida su paquete eliminando una tabla de la que depende, el paquete se revincula de forma dinámica. En este caso, se devuelve un SQLCODE -501 (SQLSTATE 24501) para una sentencia FETCH o CLOSE porque el gestor de bases de datos cierra el cursor. El modo de manejar un SQLCODE -501 (SQLSTATE 24501) en esta situación depende de si desea captar filas desde el cursor:

- Si desea captar filas desde el cursor, abra el cursor y luego ejecute la sentencia FETCH. Sin embargo, tenga en cuenta que la sentencia OPEN

vuelve a colocar el cursor al principio. La posición anterior retenida en la sentencia COMMIT WORK se pierde.

- Si no desea captar filas desde el cursor, no emita más peticiones SQL contra el cursor.

Opción WITH RELEASE

Cuando una aplicación cierra un cursor utilizando la opción WITH RELEASE, DB2® intenta liberar todos los bloqueos de lectura (READ) que el cursor sigue reteniendo. El cursor sólo continuará reteniendo bloqueos de grabación (WRITE). Si la aplicación cierra el cursor sin utilizar la opción RELEASE, los bloqueos READ y WRITE se liberarán cuando finalice la unidad de trabajo.

Tareas relacionadas:

- "Selección de varias filas utilizando un cursor" en la página 97
- "Declaración y utilización de cursores en programas de SQL estático" en la página 97

Ejemplo de un cursor en un programa de SQL estático

Los ejemplos **tut_read.sqc** en C, **tut_read.sqC/sqx** en C++, **TutRead.sqlj** en SQLJ y **cursor.sqb** en COBOL muestran cómo declarar un cursor, abrir el cursor, captar filas de la tabla y cerrar el cursor.

Puesto que REXX no da soporte a SQL estático, no se proporciona ningún ejemplo.

- C/C++

  El ejemplo tut_read muestra una sentencia select básica de una tabla que utiliza un cursor. Por ejemplo:

```
   /* declarar cursor */
    EXEC SQL DECLARE c1 CURSOR FOR
    SELECT deptnumb, deptname FROM org WHERE deptnumb < 40;

/* abrir cursor */
    EXEC SQL OPEN c1 ;

/* captar cursor */
  EXEC SQL FETCH c1 INTO :deptnumb, :deptname;

while (sqlca.sqlcode != 100)
  {
    printf("    %8d %-14s\n", deptnumb, deptname);
    EXEC SQL FETCH c1 INTO :deptnumb, :deptname;
  }

  /* cerrar cursor */
  EXEC SQL CLOSE c1;
```

- Java™

  El ejemplo **TutRead** muestra cómo leer datos de una tabla con una sencilla sentencia select utilizando un cursor. Por ejemplo:

```
// definición del cursor
  #sql iterator TutRead_Cursor(int, String);

  // declarar cursor
  TutRead_Cursor cur2;
  #sql cur2 = {SELECT deptnumb, deptname FROM org WHERE deptnumb < 40};

  // captar cursor
  #sql {FETCH :cur2 INTO :deptnumb, :deptname};

  // recuperar y mostrar el resultado de la sentencia SELECT
```

```
while (!cur2.endFetch())
{
  System.out.println(deptnumb + ", " + deptname);
  #sql {FETCH :cur2 INTO :deptnumb, :deptname};
}

// cerrar cursor
cur2.close();
```

- COBOL

  El ejemplo **cursor** muestra un ejemplo de cómo recuperar datos de una tabla utilizando un cursor con una sentencia de SQL estático. Por ejemplo:

```
* Declarar un cursor
     EXEC SQL DECLARE c1 CURSOR FOR
              SELECT name, dept FROM staff
              WHERE job='Mgr' END-EXEC.

* Abrir el cursor
     EXEC SQL OPEN c1 END-EXEC.

* Captar filas de la tabla 'staff'
     perform Fetch-Loop thru End-Fetch-Loop
        until SQLCODE not equal 0.

* Cerrar el cursor
     EXEC SQL CLOSE c1 END-EXEC.
     move "CLOSE CURSOR" to errloc.
```

Conceptos relacionados:

- "Consideraciones sobre tipos de cursor y unidad de trabajo" en la página 98
- "Recuperación de mensajes de error en una aplicación" en la página 112

Tareas relacionadas:

- "Selección de varias filas utilizando un cursor" en la página 97
- "Declaración y utilización de cursores en programas de SQL estático" en la página 97

Información relacionada:

- "Tipos de cursor" en la página 102

Ejemplos relacionados:

- "cursor.sqb -- How to update table data with cursor statically (IBM COBOL)"
- "tut_read.out -- HOW TO READ TABLES (C)"
- "tut_read.sqc -- How to read tables (C)"
- "tut_read.out -- HOW TO READ TABLES (C++)"
- "tut_read.sqC -- How to read tables (C++)"
- "TutRead.out -- HOW TO READ TABLE DATA. Connect to 'sample' database using JDBC type 2 driver (SQLJ)"
- "TutRead.sqlj -- Read data in a table (SQLj)"

Manipulación de datos recuperados

Las secciones siguientes describen cómo actualizar y suprimir datos recuperados. Los programas de ejemplo que muestran cómo manipular datos también se describen brevemente.

Actualización y supresión de datos recuperados en programas de SQL estático

Se puede actualizar y suprimir la fila a la que hace referencia un cursor. Para que una fila se pueda actualizar, la consulta correspondiente al cursor no debe ser de sólo lectura.

Procedimiento:

Para actualizar con un cursor, utilice la cláusula WHERE CURRENT OF en una sentencia UPDATE. Utilice la cláusula FOR UPDATE para indicar al sistema que desea actualizar algunas columnas de la tabla de resultados. Puede especificar una columna en la cláusula FOR UPDATE sin que esté en fullselect; por lo tanto, puede actualizar columnas que no recupera explícitamente el cursor. Si la cláusula FOR UPDATE se especifica sin nombres de columna, se considera que todas las columnas de la tabla o vista identificada en la primera cláusula FROM de fullselect externo se pueden actualizar. No nombre más columnas de las que necesita en la cláusula FOR UPDATE. En algunos casos, si se nombran columnas adicionales en la cláusula FOR UPDATE, es posible que DB2 sea menos eficiente al acceder a los datos.

La supresión con un cursor se realiza utilizando la cláusula WHERE CURRENT OF en una sentencia DELETE. En general, la cláusula FOR UPDATE no se necesita para la supresión de la fila actual de un cursor. La única excepción se produce cuando se utiliza SQL dinámico para la sentencia SELECT o la sentencia DELETE en una aplicación que se ha precompilado con el valor SAA1 para LANGLEVEL y se ha vinculado con BLOCKING ALL. En este caso, se necesita una cláusula FOR UPDATE en la sentencia SELECT.

La sentencia DELETE hace que la fila a la que hace referencia el cursor se suprima. Esta supresión deja el cursor colocado antes de la *siguiente* fila y se debe emitir una sentencia FETCH antes de que se puedan realizar operaciones WHERE CURRENT OF adicionales contra el cursor.

Información relacionada:

- "Mandato PRECOMPILE" en la publicación *Consulta de mandatos*
- "Consultas de SQL" en la publicación *Consulta de SQL, Volumen 1*

Tipos de cursor

Los cursores se dividen en tres categorías:

Sólo lectura
: Las filas del cursor sólo se pueden leer, no actualizar. Los cursores de sólo lectura se utilizan cuando una aplicación sólo va a leer datos, no a modificarlos. Un cursor se considera de sólo lectura si se basa en una sentencia select de sólo lectura. Consulte la descripción sobre cómo actualizar y recuperar datos correspondientes a normas para sentencias select que definen tablas de resultados no actualizables.

Puede haber ventajas en cuanto a rendimiento para cursores de sólo lectura.

Actualizables
Las filas del cursor se pueden actualizar. Los cursores actualizables se utilizan cuando una aplicación modifica datos a medida que se captan las filas en el cursor. La consulta especificada sólo puede hacer referencia a una tabla o vista. La consulta también debe incluir la cláusula FOR UPDATE, nombrando cada columna que se va a actualizar (a no ser que se utilice la opción de precompilación LANGLEVEL MIA).

Ambiguos
No se puede determinar si el cursor es actualizable o de sólo lectura a partir de su definición o contexto. Esta situación puede producirse cuando se encuentra una sentencia de SQL dinámico que se podría utilizar para cambiar un cursor que de otro modo se consideraría de sólo lectura.

Un cursor ambiguo se trata como de sólo lectura si se especifica la opción BLOCKING ALL al precompilar o vincular. De lo contrario, el cursor se considera actualizable.

Nota: Los cursores que se procesan de forma dinámica siempre son ambiguos.

Conceptos relacionados:

- "Modalidades de cursor soportadas en IBM OLE DB Provider" en la página 245

Tareas relacionadas:

- "Actualización y supresión de datos recuperados en programas de SQL estático" en la página 102

Ejemplo de captación en un programa de SQL estático

En el siguiente ejemplo se efectúa una selección en una tabla utilizando un cursor, se abre el cursor y se captan filas de la tabla. Para cada fila captada, el programa decide, según criterios sencillos, si la fila se debe suprimir o actualizar.

El lenguaje REXX no da soporte a SQL estático, así que no se proporciona ningún ejemplo.

- C/C++ (**tut_mod.sqc/tut_mod.sqC**)

  El siguiente ejemplo procede del ejemplo **tut_mod**. En este ejemplo se efectúa una selección en una tabla utilizando un cursor, se abre el cursor, se captan, se actualizan o se suprimen filas de la tabla y luego se cierra el cursor.

```
EXEC SQL DECLARE c1 CURSOR FOR SELECT * FROM staff WHERE id >= 310;
  EXEC SQL OPEN c1;
  EXEC SQL FETCH c1 INTO :id, :name, :dept, :job:jobInd, :years:yearsInd, :salary,
     :comm:commInd;
```

  El ejemplo **tbmod** es una versión más larga del ejemplo **tut_mod** y muestra casi todos los casos posibles de modificación de datos de la tabla.

- Java™ (**TutMod.sqlj**)

  El siguiente ejemplo procede del ejemplo **TutMod**. En este ejemplo se efectúa una selección en una tabla utilizando un cursor, se abre el cursor, se captan, se actualizan o se suprimen filas de la tabla y luego cierra el cursor.

```
#sql cur = {SELECT * FROM staff WHERE id >= 310};
#sql {FETCH :cur INTO :id, :name, :dept, :job, :years, :salary, :comm};
```

El ejemplo **TbMod** es una versión más larga del ejemplo **TutMod** y muestra casi todos los casos posibles de modificación de datos de la tabla.

- COBOL (**openftch.sqb**)

  El siguiente ejemplo procede del ejemplo **openftch**. En este ejemplo se efectúa una selección en una tabla utilizando un cursor, se abre el cursor y se captan filas de la tabla.

```
EXEC SQL DECLARE c1 CURSOR FOR
  SELECT name, dept FROM staff
  WHERE job='Mgr'
  FOR UPDATE OF job END-EXEC.

EXEC SQL OPEN c1 END-EXEC


* llamar a FETCH y al bucle UPDATE/DELETE.
  perform Fetch-Loop thru End-Fetch-Loop
  until SQLCODE not equal 0.

EXEC SQL CLOSE c1 END-EXEC.
```

Conceptos relacionados:

- "Recuperación de mensajes de error en una aplicación" en la página 112

Ejemplos relacionados:

- "openftch.sqb -- How to modify table data using cursor statically (IBM COBOL)"
- "tbmod.sqc -- How to modify table data (C)"
- "tut_mod.out -- HOW TO MODIFY TABLE DATA (C)"
- "tut_mod.sqc -- How to modify table data (C)"
- "tbmod.sqC -- How to modify table data (C++)"
- "tut_mod.out -- HOW TO MODIFY TABLE DATA (C++)"
- "tut_mod.sqC -- How to modify table data (C++)"
- "TbMod.sqlj -- How to modify table data (SQLj)"
- "TutMod.out -- HOW TO MODIFY TABLE DATA. Connect to 'sample' database using JDBC type 2 driver (SQLJ)"
- "TutMod.sqlj -- Modify data in a table (SQLj)"

Desplazamiento por datos recuperados y manipulación de los mismos

Las secciones siguientes describen cómo desplazarse por datos recuperados. Los programas de ejemplo que muestran cómo manipular datos también se describen brevemente.

Desplazamiento por datos recuperados previamente

Cuando una aplicación recupera datos de la base de datos, la sentencia FETCH le permite desplazarse hacia adelante por los datos; sin embargo, el gestor de bases de datos no tiene ninguna sentencia de SQL incorporado que le permita desplazarse hacia atrás por los datos (equivalente a una operación FETCH hacia atrás). Sin embargo, las CLI de DB2 y Java dan soporte a una operación FETCH hacia atrás por cursores desplazables de sólo lectura.

Procedimiento:

Para aplicaciones de SQL incorporado, puede utilizar las siguientes técnicas para desplazarse por los datos que se han recuperado:

- Conserve una copia de los datos que se han captado y desplácese por los mismos mediante alguna técnica de programación.
- Utilice SQL para recuperar de nuevo los datos, normalmente mediante una segunda sentencia SELECT.

Tareas relacionadas:

- "Cómo conservar una copia de los datos" en la página 105
- "Recuperación de datos por segunda vez" en la página 105

Información relacionada:

- "SQLFetchScroll function (CLI) - Fetch rowset and return data for all bound columns" en la publicación *CLI Guide and Reference, Volume 2*
- "Cursor positioning rules for SQLFetchScroll() (CLI)" en la publicación *CLI Guide and Reference, Volume 2*

Cómo conservar una copia de los datos

En algunas situaciones, puede resultar útil mantener una copia de los datos que capta la aplicación.

Procedimiento:

Para conservar una copia de los datos, la aplicación puede hacer lo siguiente:

- Guardar los datos captados en almacenamiento virtual.
- Grabar los datos en un archivo temporal (si los datos no caben en almacenamiento virtual). Un efecto de este enfoque es que un usuario que se desplace hacia atrás siempre ve exactamente los mismos datos que se han captado, incluso si mientras tanto una transacción ha modificado los datos de la base de datos.
- Utilizando un nivel de aislamiento de lectura repetida, los datos que recupera de una transacción se pueden volver a recuperar cerrando y abriendo un cursor. Otras aplicaciones no pueden actualizar los datos del conjunto de resultados. Los niveles de aislamiento y el bloqueo pueden afectar al modo en que los usuarios actualizan datos.

Conceptos relacionados:

- "Diferencias en el orden de filas entre la primera y la segunda tabla de resultados" en la página 106

Tareas relacionadas:

- "Recuperación de datos por segunda vez" en la página 105

Recuperación de datos por segunda vez

La técnica que utilice para recuperar datos por segunda vez dependerá del orden en el que desee volver a ver los datos.

Procedimiento:

Puede recuperar datos por segunda vez mediante cualquiera de los métodos siguientes:

- Recuperar datos desde el principio

  Para volver a recuperar los datos desde el principio de la tabla de resultados, cierre el cursor activo y vuélvalo a abrir. Esta acción coloca el cursor al principio de la tabla de resultados. Pero, a no ser que la aplicación retenga bloqueos en la tabla, es posible que otros la hayan modificado, de modo que puede que la que era la primera fila de la tabla de resultados ya no lo sea.
- Recuperar datos desde el medio

  Para recuperar datos por segunda vez desde algún punto del medio de la tabla de resultados, ejecute una segunda sentencia SELECT y declare un segundo cursor en la sentencia. Por ejemplo, supongamos que la primera sentencia SELECT era la siguiente:

  ```
  SELECT * FROM DEPARTMENT
    WHERE LOCATION = 'CALIFORNIA'
    ORDER BY DEPTNO
  ```

  Ahora, supongamos que desea volver a las filas que empiezan con `DEPTNO = 'M95'` y captar de forma secuencial desde dicho punto. Codifique lo siguiente:

  ```
  SELECT * FROM DEPARTMENT
    WHERE LOCATION = 'CALIFORNIA'
    AND DEPTNO >= 'M95'
    ORDER BY DEPTNO
  ```

  Esta sentencia coloca el cursor donde desea.
- Recuperar datos en orden inverso

  El valor por omisión consiste en ordenar las filas en orden ascendente. Si sólo hay una fila para cada valor de `DEPTNO`, la siguiente sentencia especifica un orden ascendente exclusivo de filas:

  ```
  SELECT * FROM DEPARTMENT
    WHERE LOCATION = 'CALIFORNIA'
    ORDER BY DEPTNO
  ```

  Para recuperar las mismas filas en orden inverso, especifique que el orden debe ser descendente, como en la siguiente sentencia:

  ```
  SELECT * FROM DEPARTMENT
    WHERE LOCATION = 'CALIFORNIA'
    ORDER BY DEPTNO DESC
  ```

  Un cursor de la segunda sentencia recupera filas exactamente en el orden inverso al de un cursor de la primera sentencia. El orden de recuperación sólo se garantiza si la primera sentencia especifica una secuencia de orden exclusiva.

  Para recuperar filas en orden inverso, puede resultar útil tener dos índices en la columna `DEPTNO`, uno en orden ascendente y el otro en orden descendente.

Conceptos relacionados:

- "Diferencias en el orden de filas entre la primera y la segunda tabla de resultados" en la página 106

Diferencias en el orden de filas entre la primera y la segunda tabla de resultados

Puede que las filas de la segunda tabla de resultados no se visualicen en el mismo orden que en la primera. El gestor de bases de datos no tiene en cuenta el orden de las filas como algo significativo a no ser que la sentencia SELECT utilice ORDER BY. Por lo tanto, si hay varias filas con el mismo valor `DEPTNO`, puede que la segunda sentencia SELECT las recupere en un orden distinto al de la primera. La única garantía es que todas estarán en orden por número de departamento, tal como solicita la cláusula ORDER BY DEPTNO.

La diferencia en el orden se puede producir aunque ejecute la misma sentencia de SQL, con las mismas variables del lenguaje principal, por segunda vez. Por ejemplo, las estadísticas del catálogo se podrían haber actualizado entre ejecuciones, o se podrían hacer creado o eliminado índices. Entonces podría ejecutar de nuevo la sentencia SELECT.

Es más probable que el orden cambie si la segunda sentencia SELECT tiene un predicado que la primera no tenía; el gestor de bases de datos podría elegir utilizar un índice en el nuevo predicado. Por ejemplo, podría elegir un índice en LOCATION para la primera sentencia de nuestro ejemplo y un índice en DEPTNO para la segunda. Puesto que las filas se captan en orden por la clave de índice, el segundo orden no es necesariamente igual al primero.

De nuevo, al ejecutar dos sentencias SELECT parecidas se puede producir un orden diferente de filas, aunque no se haya producido ningún cambio en las estadísticas ni se haya creado ni eliminado ningún índice. En el ejemplo, si hay varios valores diferentes de LOCATION, el gestor de bases de datos podría elegir un índice en LOCATION para ambas sentencias. Si se cambia el valor de DEPTNO en la segunda sentencia por lo siguiente, el gestor de bases de datos podría elegir un índice en DEPTNO:

```
SELECT * FROM DEPARTMENT
  WHERE LOCATION = 'CALIFORNIA'
  AND DEPTNO >= 'Z98'
  ORDER BY DEPTNO
```

Debido a la sutil relación entre el formato de una sentencia de SQL y los valores de dicha sentencia, nunca dé por supuesto que dos sentencias de SQL diferentes vayan a devolver filas en el mismo orden, a no ser que el orden se determine de forma exclusiva mediante una cláusula ORDER BY.

Tareas relacionadas:

• "Recuperación de datos por segunda vez" en la página 105

Colocación de un cursor al final de una tabla

Si tiene que colocar el cursor al final de una tabla, puede utilizar una sentencia de SQL para colocarlo.

Procedimiento:

Utilice cualquiera de los ejemplos siguientes como método para colocar un cursor:

- El gestor de bases de datos no garantiza un orden en los datos almacenados en una tabla; por lo tanto, el final de una tabla no está definido. Sin embargo, el orden se define en el resultado de una sentencia de SQL:

  ```
  SELECT * FROM DEPARTMENT
    ORDER BY DEPTNO DESC
  ```

- La sentencia siguiente coloca el cursor en la fila que tiene el valor de DEPTNO más alto:

  ```
  SELECT * FROM DEPARTMENT
    WHERE DEPTNO =
    (SELECT MAX(DEPTNO) FROM DEPARTMENT)
  ```

Sin embargo, observe que, si hay varias filas con el mismo valor, el cursor se coloca en la primera de ellas.

Actualización de datos recuperados previamente

Para desplazarse hacia atrás y actualizar datos recuperados previamente, puede utilizar una combinación de las técnicas que se utilizan para desplazarse a través de datos recuperados previamente y para actualizar datos recuperados.

Procedimiento:

Para actualizar datos recuperados previamente, puede hacer una de estas dos cosas:

- Si tiene un segundo cursor en los datos que se tienen que actualizar y la sentencia SELECT no utiliza ninguno de los elementos restringidos, puede utilizar la sentencia UPDATE controlada por el cursor. Nombre el segundo cursor en la cláusula WHERE CURRENT OF.
- En otros casos, utilice UPDATE con una cláusula WHERE que nombre todos los valores de la fila o especifique la clave primaria de la tabla. Puede ejecutar una sentencia varias veces con distintos valores de las variables.

Tareas relacionadas:

- "Actualización y supresión de datos recuperados en programas de SQL estático" en la página 102
- "Desplazamiento por datos recuperados previamente" en la página 104

Ejemplo de inserción, actualización y supresión en un programa de SQL estático

Los siguientes ejemplos muestran cómo insertar, actualizar y suprimir datos mediante SQL estático.

- C/C++ (**tut_mod.sqc/tut_mod.sqC**)

  Los tres ejemplos siguientes proceden del ejemplo **tut_mod**. Consulte este ejemplo para ver un programa completo que muestra cómo modificar datos de tablas en C o C++.

  El siguiente ejemplo muestra cómo insertar datos de tablas:

```
EXEC SQL INSERT INTO staff(id, name, dept, job, salary)
  VALUES(380, 'Pearce', 38, 'Clerk', 13217.50),
        (390, 'Hachey', 38, 'Mgr', 21270.00),
        (400, 'Wagland', 38, 'Clerk', 14575.00);
```

  El siguiente ejemplo muestra cómo actualizar datos de tablas:

```
EXEC SQL UPDATE staff
  SET salary = salary + 10000
  WHERE id >= 310 AND dept = 84;
```

  El siguiente ejemplo muestra cómo suprimir datos de una tabla:

```
EXEC SQL DELETE
  FROM staff
  WHERE id >= 310 AND salary > 20000;
```

- Java™ (**TutMod.sqlj**)

  Los tres ejemplos siguientes proceden del ejemplo **TutMod**. Consulte este ejemplo para ver un programa completo que muestra cómo modificar datos de tablas en SQLJ.

  El siguiente ejemplo muestra cómo insertar datos de tablas:

```
#sql {INSERT INTO staff(id, name, dept, job, salary)
        VALUES(380, 'Pearce', 38, 'Clerk', 13217.50),
              (390, 'Hachey', 38, 'Mgr', 21270.00),
              (400, 'Wagland', 38, 'Clerk', 14575.00)};
```

El siguiente ejemplo muestra cómo actualizar datos de tablas:

```
#sql {UPDATE staff
        SET salary = salary + 1000
        WHERE id >= 310 AND dept = 84};
```

El siguiente ejemplo muestra cómo suprimir datos de una tabla:

```
#sql {DELETE FROM staff
        WHERE id >= 310 AND salary > 20000};
```

- COBOL (**updat.sqb**)

  Los tres ejemplos siguientes proceden del ejemplo **updat**. Consulte este ejemplo para ver un programa completo que muestra cómo modificar datos de tablas en COBOL.

  El siguiente ejemplo muestra cómo insertar datos de tablas:

```
EXEC SQL INSERT INTO staff
    VALUES (999, 'Testing', 99, :job-update, 0, 0, 0)
    END-EXEC.
```

  El siguiente ejemplo muestra cómo actualizar datos de tablas:

```
EXEC SQL UPDATE staff
    SET job=:job-update
    WHERE job='Mgr'
    END-EXEC.
```

  El siguiente ejemplo muestra cómo suprimir datos de una tabla:

```
EXEC SQL DELETE
    FROM staff
    WHERE job=:job-update
    END-EXEC.
```

Conceptos relacionados:

- “Recuperación de mensajes de error en una aplicación” en la página 112

Ejemplos relacionados:

- “tbinfo.out -- HOW TO GET INFORMATION AT THE TABLE LEVEL (C++)”
- “tbmod.out -- HOW TO MODIFY TABLE DATA (C++)”
- “tbmod.sqC -- How to modify table data (C++)”
- “tut_mod.out -- HOW TO MODIFY TABLE DATA (C++)”
- “tut_mod.sqC -- How to modify table data (C++)”
- “tbmod.out -- HOW TO MODIFY TABLE DATA (C)”
- “tbmod.sqc -- How to modify table data (C)”
- “tut_mod.out -- HOW TO MODIFY TABLE DATA (C)”
- “tut_mod.sqc -- How to modify table data (C)”
- “TbMod.out -- HOW TO MODIFY TABLE DATA. Connect to ’sample’ database using JDBC type 2 driver (SQLJ)”
- “TbMod.sqlj -- How to modify table data (SQLj)”
- “TutMod.out -- HOW TO MODIFY TABLE DATA. Connect to ’sample’ database using JDBC type 2 driver (SQLJ)”
- “TutMod.sqlj -- Modify data in a table (SQLj)”

Información de diagnóstico

Las secciones siguientes describen la información de diagnóstico disponible para un programa de SQL estático, como códigos de retorno, y cómo debe la aplicación recuperar mensajes de error.

Códigos de retorno

La mayoría de las API del gestor de bases de datos devuelven un código de retorno cero cuando se ejecutan satisfactoriamente. En general, un código de retorno distinto de cero indica que el mecanismo de manejo de errores secundarios, la estructura SQLCA, puede estar dañado. En este caso, la API llamada no se ejecuta. Una posible razón de que se dañe la estructura SQLCA es que se haya pasado una dirección no válida para la estructura.

Información relacionada:

- "SQLCA" en la publicación *Administrative API Reference*

Información de error en los campos SQLCODE, SQLSTATE y SQLWARN

La información de error se devuelve en los campos SQLCODE y SQLSTATE de la estructura SQLCA, que se actualiza tras cada sentencia de SQL ejecutable y tras la mayoría de las llamadas a API del gestor de bases de datos.

Un archivo fuente que contiene sentencias de SQL ejecutables puede proporcionar al menos una estructura SQLCA con el nombre `sqlca`. La estructura SQLCA se define en el archivo include SQLCA. Los archivos fuente sin sentencias de SQL incorporado, pero que llaman a las API del gestor de bases de datos, también pueden proporcionar una o más estructuras SQLCA, pero sus nombres son arbitrarios.

Si la aplicación cumple con el estándar FIPS 127-2, puede declarar SQLSTATE y SQLCODE como variables del lenguaje principal para aplicaciones C, C++, COBOL y FORTRAN, en lugar de utilizar la estructura SQLCA.

Un valor SQLCODE igual a 0 significa una ejecución satisfactoria (con posibles condiciones de aviso SQLWARN). Un valor positivo significa que la sentencia se ha ejecutado satisfactoriamente pero con un aviso, como el truncamiento de una variable del lenguaje principal. Un valor negativo significa que se ha producido una condición de error.

Un campo adicional, SQLSTATE, contiene un código de error estandarizado coherente con otros productos de bases de datos de IBM® y con otros gestores de bases de datos que cumplen con SQL92. En la práctica, es conveniente utilizar valores SQLSTATE cuando le preocupe la portabilidad, pues los valores SQLSTATE son utilizados por muchos gestores de bases de datos.

El campo SQLWARN contiene una matriz de indicadores de aviso, incluso si SQLCODE es cero. El primer elemento de la matriz SQLWARN, SQLWARN0, contiene un blanco si todos los demás elementos están en blanco. SQLWARN0 contiene una `W` si al menos uno de los otros elementos contiene un carácter de aviso.

Nota: Si desea desarrollar aplicaciones que acceden a varios servidores RDBMS de IBM, debe:

- Si es posible, hacer que las aplicaciones comprueben SQLSTATE en lugar de SQLCODE.
- Si la aplicación va a utilizar DB2 Connect, considerar la posibilidad de utilizar el recurso de correlación que proporciona DB2 Connect para correlacionar conversiones de SQLCODE entre bases de datos distintas.

Conceptos relacionados:

- "Códigos de retorno" en la página 110
- "Variables SQLSTATE y SQLCODE en C y C++" en la página 186
- "Variables SQLSTATE y SQLCODE en COBOL" en la página 213
- "Variables SQLSTATE y SQLCODE en FORTRAN" en la página 229
- "Variables SQLSTATE y SQLCODE en Perl" en la página 527

Información relacionada:

- "SQLCA" en la publicación *Administrative API Reference*

Truncamiento de símbolos en la estructura SQLCA

Puesto que los símbolos se pueden truncar en la estructura SQLCA, no debe utilizar la información de símbolos cuando realice diagnósticos. Aunque puede definir nombres de tabla y de columna con longitudes de hasta 128 bytes, los símbolos de SQLCA se truncarán a 17 bytes más un terminador de truncamiento (>). La lógica de la aplicación no debería depender de los valores reales del campo `sqlerrmc`.

Información relacionada:

- "SQLCA" en la publicación *Administrative API Reference*

Consideraciones sobre el manejador de excepciones, señales e interrupciones

Un manejador de excepciones, señales o interrupciones es una rutina que adquiere el control cuando se produce una excepción, señal o interrupción. El tipo de manejador aplicable lo determina el entorno operativo, tal como se muestra a continuación:

Sistemas operativos Windows®
: Al pulsar `Control-C` o `Control-Inter` se genera una interrupción.

Sistemas basados en UNIX®
: Generalmente, al pulsar `Control-C` se genera una señal de interrupción SIGINT. Observe que los teclados se pueden redefinir fácilmente de modo que se pueda generar una señal SIGINT pulsando otra secuencia de teclas en la máquina.

No coloque sentencias de SQL (que no sean COMMIT o ROLLBACK) en manejadores de excepciones, señales e interrupciones. Con estos tipos de condiciones de error, generalmente deseará llevar a cabo una operación ROLLBACK para evitar el riesgo de tener datos incoherentes.

Tenga en cuenta que debe tener cuidado cuando codifique COMMIT y ROLLBACK en manejadores de excepciones/señales/interrupciones. Si llama a cualquiera de

estas sentencias por ellas mismas, la operación COMMIT o ROLLBACK no se ejecuta hasta que se completa la sentencia de SQL actual, si se está ejecutando alguna. Este no es el comportamiento deseado de un manejador `Control-C`.

La solución consiste en llamar a la API INTERRUPT (`sqleintr/sqlgintr`) antes de emitir una operación ROLLBACK. Esta API interrumpe la consulta de SQL actual (si la aplicación está ejecutando alguna) y deja que ROLLBACK comience de inmediato. Si va a realizar una operación COMMIT en lugar de ROLLBACK, no desea interrumpir el mandato actual.

Cuando se utiliza APPC para acceder a un servidor de bases de datos remoto (DB2 para AIX o el sistema de bases de datos de sistema principal que utiliza DB2 Connect), es posible que la aplicación reciba una señal SIGUSR1. SNA Services/6000 genera esta señal cuando se produce un error no recuperable y la conexión SNA se detiene. Es posible que desee instalar un manejador de señales en la aplicación para que maneje SIGUSR1.

Consulte la documentación de su plataforma para ver detalles específicos sobre consideraciones sobre distintos manejadores.

Conceptos relacionados:

• "Proceso de peticiones de interrupción" en la página 743

Consideraciones sobre las rutinas de lista de salida

No utilice SQL ni llamadas a API de DB2 en rutinas de lista de salida. Tenga en cuenta que no puede desconectarse de una base de datos en una rutina de salida.

Recuperación de mensajes de error en una aplicación

En función del lenguaje en el que esté escrita la aplicación, debe utilizar un método u otro para recuperar información de error:

- Las aplicaciones C, C++ y COBOL pueden utilizar la API GET ERROR MESSAGE para obtener la información correspondiente relacionada con el SQLCA que se ha pasado.
- Las aplicaciones JDBC y SQLJ emiten una SQLException cuando se produce un error durante el proceso de SQL. Las aplicaciones pueden obtener y visualizar una SQLException con el siguiente código:

```
try {
   Statement stmt = connection.createStatement();
   int rowsDeleted = stmt.executeUpdate(
      "DELETE FROM employee WHERE empno = '000010'");
   System.out.println( rowsDeleted + " rows were deleted");
}

catch (SQLException sqle) {
   System.out.println(sqle);
}
```

- Las aplicaciones REXX utilizan el procedimiento CHECKERR.

Conceptos relacionados:

• "Variables SQLSTATE y SQLCODE en C y C++" en la página 186
• "Variables SQLSTATE y SQLCODE en COBOL" en la página 213
• "Variables SQLSTATE y SQLCODE en FORTRAN" en la página 229
• "Variables SQLSTATE y SQLCODE en Perl" en la página 527

Información relacionada:

- "sqlaintp - Get Error Message" en la publicación *Administrative API Reference*

Capítulo 5. Cómo escribir programas de SQL dinámico

Características y razones para utilizar SQL dinámico 115
Razones para utilizar SQL dinámico 115
Sentencias de soporte de SQL dinámico . . . 116
SQL dinámico frente a SQL estático 116
Cursores en programas de SQL dinámico 119
Declaración y utilización de cursores en programas de SQL dinámico 119
Ejemplo de un cursor en un programa de SQL dinámico 120
| Efectos de REOPT en el SQL dinámico 121
Efecto de la opción de vinculación DYNAMICRULES en SQL dinámico. 122
La SQLDA en programas de SQL dinámico . . . 124
Variables del lenguaje principal en la SQLDA en programas de SQL dinámico 124
Declaración de la estructura SQLDA en un programa de SQL dinámico 125
Preparación de una sentencia de SQL dinámico utilizando la estructura SQLDA mínima . . . 126
Asignación de una SQLDA con suficientes entradas SQLVAR para un programa de SQL dinámico 128
Descripción de una sentencia SELECT en un programa de SQL dinámico 129
Adquisición de almacenamiento para albergar una fila 129
Proceso del cursor en un programa de SQL dinámico 130
Asignación de una estructura SQLDA para un programa de SQL dinámico 131
Transferencia de datos en un programa de SQL dinámico mediante una estructura SQLDA . . 134
Proceso de sentencias interactivas de SQL en programas de SQL dinámico 135
Determinación del tipo se sentencia en programas de SQL dinámico 135
Proceso de sentencias SELECT de lista de variables en programas de SQL dinámico . . . 136
Cómo guardar peticiones de SQL procedentes de usuarios finales. 137
Marcadores de parámetros en programas de SQL dinámico 137
Cómo proporcionar entrada de variables a SQL dinámico mediante marcadores de parámetros . 137
Ejemplo de marcadores de parámetros en un programa de SQL dinámico 138
Comparación entre Interfaz de nivel de llamada (CLI) de DB2 y SQL dinámico 140
La Interfaz de nivel de llamada (CLI) de DB2 y el SQL dinámico incorporado 140
Ventajas de CLI de DB2 sobre el SQL incorporado 141
Cuándo utilizar CLI de DB2 o SQL incorporado 143

Características y razones para utilizar SQL dinámico

Las secciones siguientes describen las razones para utilizar SQL dinámico, en comparación con SQL estático.

Razones para utilizar SQL dinámico

Es posible que desee utilizar SQL dinámico si:

- Necesita que toda la sentencia de SQL, o parte de la misma, se genere durante la ejecución de la aplicación.
- Los objetos a los que hace referencia la sentencia de SQL no existen en el momento de la precompilación.
- Desea que la sentencia siempre utilice la vía de acceso óptima, según las estadísticas actuales de la base de datos.
- Desea modificar el entorno de compilación de la sentencia, es decir, experimentar con los registros especiales.

Conceptos relacionados:

- "Sentencias de soporte de SQL dinámico" en la página 116
- "SQL dinámico frente a SQL estático" en la página 116

Sentencias de soporte de SQL dinámico

Las sentencias de soporte de SQL dinámico aceptan una variable del lenguaje principal de serie de caracteres y un nombre de sentencia como argumentos. La variable del lenguaje principal contiene la sentencia de SQL que se va a procesar de forma dinámica en formato de texto. El texto de la sentencia no se procesa cuando se precompila una aplicación. De hecho, el texto de la sentencia no tiene que existir en el momento en que se precompila la aplicación. En su lugar, la sentencia de SQL se trata como una variable del lenguaje principal con fines de precompilación y se hace referencia a la variable durante la ejecución de la aplicación. Estas sentencias de SQL se denominan SQL *dinámico*.

Se necesitan sentencias de soporte de SQL dinámico para transformar la variable del lenguaje principal que contiene texto de SQL en un formato ejecutable y trabajar en el mismo haciendo referencia al nombre de la sentencia. Estas sentencias son:

EXECUTE IMMEDIATE
Prepara y ejecuta una sentencia que no utiliza ninguna variable del lenguaje principal. Todas las sentencias EXECUTE IMMEDIATE de una aplicación se colocan en antememoria en el mismo lugar en el tiempo de ejecución, de modo que sólo se conoce la última sentencia. Utilice esta sentencia como alternativa a las sentencias PREPARE y EXECUTE.

PREPARE
Convierte el formato de serie de caracteres de la sentencia de SQL en un formato ejecutable de la sentencia, asigna un nombre de sentencia y opcionalmente coloca información sobre la sentencia en una estructura SQLDA.

EXECUTE
Ejecuta una sentencia de SQL previamente preparada. La sentencia se puede ejecutar repetidamente dentro de una conexión.

DESCRIBE
Coloca información sobre una sentencia preparada en una SQLDA.

Una aplicación puede ejecutar la mayoría de las sentencias de SQL soportadas de forma dinámica.

Nota: El contenido de las sentencias de SQL dinámico siguen la misma sintaxis que las sentencias de SQL estático, con las siguientes excepciones:

- No se permiten comentarios.
- La sentencia no puede comenzar por EXEC SQL.
- La sentencia no puede finalizar con el terminador de sentencia. Una excepción a esta norma es la sentencia CREATE TRIGGER, que puede contener un signo de punto y coma (;).

Información relacionada:

• Apéndice A, “Sentencias de SQL soportadas”, en la página 733

SQL dinámico frente a SQL estático

La pregunta sobre si utilizar SQL estático o dinámico para optimizar el rendimiento suele resultar de gran interés para los programadores. La respuesta depende de la situación.

Utilice la tabla siguiente para decidir si utilizar SQL estático o dinámico. Consideraciones como seguridad indican utilizar SQL estático, mientras que consideraciones de entorno (por ejemplo, utilizar CLI de DB2 o el CLP) indican utilizar SQL dinámico. Cuando tome su decisión, tenga en cuenta las siguientes recomendaciones sobre si se debe elegir SQL estático o dinámico en una determinada situación. En la tabla siguiente, 'Cualquiera' significa que no hay ninguna ventaja en utilizar SQL estático o dinámico.

Nota: Esta tabla contiene recomendaciones generales. La aplicación específica, el uso para el que esté pensada y el entorno de trabajo dictan la opción real. Cuando tenga dudas, el mejor enfoque consiste en hacer prototipos de sus sentencias como SQL estático, luego como SQL dinámico y luego comparar las diferencias.

Tabla 8. Comparación entre SQL estático y dinámico

Consideraciones	**Probablemente la mejor opción**
Tiempo de ejecución de la sentencia de SQL:	
• Menos de 2 segundos	• Estático
• Entre 2 y 10 segundos	• Cualquiera
• Más de 10 segundos	• Dinámico
Uniformidad de datos	
• Distribución de datos uniforme	• Estático
• Ligera falta de uniformidad	• Cualquiera
• Distribución nada uniforme	• Dinámico
Predicados de rango (<,>,BETWEEN,LIKE)	
• Muy poco frecuentes	• Estático
• Ocasionales	• Cualquiera
• Frecuentes	• Dinámico
Ejecución repetitiva	
• Se ejecuta muchas veces (10 o más)	• Cualquiera
• Se ejecuta unas cuantas veces (menos de 10)	• Cualquiera
• Se ejecuta una vez	• Estático
Naturaleza de la consulta	
• Aleatoria	• Dinámico
• Permanente	• Cualquiera
Entorno de tiempo de ejecución (DML/DDL)	
• Proceso de transacciones (sólo DML)	• Cualquiera
• Mixto (DML y DDL - DDL afecta a los paquetes)	• Dinámico
• Mixto (DML y DDL - DDL no afecta a los paquetes)	• Cualquiera
Frecuencia de `RUNSTATS`	
• Muy poco frecuente	• Estático
• Regular	• Cualquiera
• Frecuente	• Dinámico

En general, una aplicación que utiliza SQL dinámico tiene un coste de partida (inicial) superior por sentencia de SQL debido a la necesidad de compilar las sentencias de SQL antes de utilizarlas. Una vez compiladas, el tiempo de ejecución de SQL dinámico comparado con SQL estático debe ser equivalente y, en algunos casos, más rápido debido a que el optimizador elige mejores planes de acceso. Cada vez que se ejecuta una sentencia dinámica, el coste de compilación inicial pierde importancia. Si varios usuarios ejecutan la misma aplicación dinámica con las mismas sentencias, el coste de compilación de la sentencia sólo se aplica a la primera aplicación que emite la sentencia.

En un entorno DML y DDL mixto, el coste de compilación correspondiente a una sentencia de SQL dinámico puede variar puesto que es posible que el sistema recompile de forma implícita la sentencia mientras se ejecuta la aplicación. En un entorno mixto, la opción entre SQL estático y dinámico debe depender también de la frecuencia con la que se invalidan paquetes. Si el DDL no invalida paquetes, es posible que SQL dinámico resulte más eficiente puesto que sólo las consultas ejecutadas se recompilan cuando se utilizan la siguiente vez. Las otras no se recompilan. Para SQL estático, el paquete entero se revincula cuando se ha invalidado.

Ahora supongamos que su aplicación particular contiene una combinación de las características anteriores y algunas de estas características sugieren que utilice SQL estático y otras que utilice SQL dinámico. En este caso, no hay ninguna decisión obvia y probablemente deba utilizar el método con el que tenga más experiencia y con el que se sienta más cómodo. Observe que las consideraciones de la tabla anterior se listan en líneas generales por orden de importancia.

Nota: Tanto SQL estático como SQL dinámico vienen en dos tipos que constituyen una diferencia para el optimizador de DB2. Estos tipos son:

1. SQL estático que no contiene variables del lenguaje principal

   Esta es una situación poco probable que sólo verá para:

   - Código de *inicialización*
   - Ejemplos de formación para principiantes

   Realmente es la mejor combinación, desde la perspectiva del rendimiento, puesto que no hay proceso general de rendimiento en tiempo de ejecución y se pueden aprovechar por completo las funciones del optimizador de DB2.
2. SQL estático que contiene variables del lenguaje principal

   Es el estilo *antiguo* tradicional de las aplicaciones de DB2®. Evita el proceso general en tiempo de ejecución de una sentencia PREPARE y bloqueos de catálogo adquiridos durante la compilación de sentencias. Desgraciadamente, no se puede utilizar la potencia completa del optimizador porque este no conoce la sentencia de SQL completa. Existe un problema particular con distribuciones de datos nada uniformes.
3. SQL dinámico que no contiene marcadores de parámetros

   Es el estilo típico para interfaces de consultas aleatorias (como el CLP) y es el *estilo* de SQL preferido del optimizador. Para consultas complejas, el proceso general de la sentencia PREPARE queda compensado por la mejora en el tiempo de ejecución.
4. SQL dinámico que contiene marcadores de parámetros

   Es el tipo más común de SQL para aplicaciones CLI. La ventaja clave es que la presencia de marcadores de parámetros permite amortizar el coste de la sentencia PREPARE durante ejecuciones repetidas de la sentencia, normalmente una sentencia SELECT o INSERT. Esta amortización es real para todas las aplicaciones de SQL dinámico repetitivas.
   Desgraciadamente, al igual que SQL estático con variables del lenguaje principal, partes del optimizador de DB2 no funcionarán porque no está disponible la información completa. La recomendación es utilizar *SQL estático con variables del lenguaje principal* o *SQL dinámico sin marcadores de parámetros* como las opciones más eficientes.

Conceptos relacionados:

- "Ejemplo de marcadores de parámetros en un programa de SQL dinámico" en la página 138

Tareas relacionadas:

- "Cómo proporcionar entrada de variables a SQL dinámico mediante marcadores de parámetros" en la página 137

Cursores en programas de SQL dinámico

Las secciones siguientes describen cómo declarar y utilizar cursores en SQL dinámico y describen brevemente los programas de ejemplo que utilizan cursores.

Declaración y utilización de cursores en programas de SQL dinámico

Procesar un cursor de forma dinámica es casi idéntico a procesarlo mediante SQL estático. Cuando se declara un cursor, se asocia con una consulta.

En SQL estático, la consulta es una sentencia SELECT en formato de texto, mientras que en SQL dinámico la consulta se asocia con un nombre de sentencia asignado en una sentencia PREPARE. Cualquier variable del lenguaje principal a la que se haga referencia se representa mediante marcadores de parámetros.

La principal diferencia entre un cursor estático y uno dinámico es que un cursor estático se prepara en el momento de la precompilación y un cursor dinámico se prepara en el tiempo de ejecución. Además, las variables del lenguaje principal a las que se hace referencia en la consulta están representadas mediante marcadores de parámetros, los cuales se sustituyen por variables del lenguaje principal de tiempo de ejecución cuando se abre el cursor.

Procedimiento:

Utilice los ejemplos que se muestran en la tabla siguiente cuando codifique cursores para un programa de SQL dinámico:

Tabla 9. Sentencia declare asociada con una sentencia SELECT dinámica

Lenguaje	Código fuente de ejemplo

C/C++	`strcpy( prep_string, "SELECT tabname FROM syscat.tables"`
	`"WHERE tabschema = ?" );`
	`EXEC SQL PREPARE s1 FROM :prep_string;`
	`EXEC SQL DECLARE c1 CURSOR FOR s1;`
	`EXEC SQL OPEN c1 USING :host_var;`

Java (JDBC)	`PreparedStatement prep_string = ("SELECT tabname FROM syscat.tables`
	`WHERE tabschema = ?" );`
	`prep_string.setCursor("c1");`
	`prep_string.setString(1, host_var);`
	`ResultSet rs = prep_string.executeQuery();`

COBOL	`MOVE "SELECT TABNAME FROM SYSCAT.TABLES WHERE TABSCHEMA = ?"`
	`TO PREP-STRING.`
	`EXEC SQL PREPARE S1 FROM :PREP-STRING END-EXEC.`
	`EXEC SQL DECLARE C1 CURSOR FOR S1 END-EXEC.`
	`EXEC SQL OPEN C1 USING :host-var END-EXEC.`

Tabla 9. Sentencia declare asociada con una sentencia SELECT dinámica (continuación)

Lenguaje	Código fuente de ejemplo

FORTRAN	`prep_string = 'SELECT tabname FROM syscat.tables WHERE tabschema = ?'`
	`EXEC SQL PREPARE s1 FROM :prep_string`
	`EXEC SQL DECLARE c1 CURSOR FOR s1`
	`EXEC SQL OPEN c1 USING :host_var`

Conceptos relacionados:

- "Ejemplo de un cursor en un programa de SQL dinámico" en la página 120
- "Cursores en REXX" en la página 539

Tareas relacionadas:

- "Selección de varias filas utilizando un cursor" en la página 97

Ejemplo de un cursor en un programa de SQL dinámico

Una sentencia de SQL dinámico se puede preparar para su ejecución con la sentencia PREPARE y se puede ejecutar con la sentencia EXECUTE o con la sentencia DECLARE CURSOR.

PREPARE con EXECUTE

El siguiente ejemplo muestra cómo se puede preparar una sentencia de SQL dinámico para su ejecución con la sentencia PREPARE y ejecutar con la sentencia EXECUTE:

- C/C++ (**dbuse.sqc/dbuse.sqC**):

  El siguiente ejemplo procede del ejemplo **dbuse**:

```
EXEC SQL BEGIN DECLARE SECTION;
  char hostVarStmt[50];
EXEC SQL END DECLARE SECTION;

strcpy(hostVarStmt, "DELETE FROM org WHERE deptnumb = 15");
EXEC SQL PREPARE Stmt FROM :hostVarStmt;
EXEC SQL EXECUTE Stmt;
```

PREPARE con DECLARE CURSOR

Los ejemplos siguientes muestran cómo se puede preparar una sentencia de SQL dinámico para su ejecución con la sentencia PREPARE y ejecutar con la sentencia DECLARE CURSOR:

- C

```
EXEC SQL BEGIN DECLARE SECTION;
  char  st[80];
  char  parm_var[19};
EXEC SQL END DECLARE SECTION;

strcpy( st, "SELECT tabname FROM syscat.tables" );
strcat( st, " WHERE tabname <> ? ORDER BY 1" );
EXEC SQL PREPARE s1 FROM :st;
EXEC SQL DECLARE c1 CURSOR FOR s1;
strcpy( parm_var, "STAFF" );
EXEC SQL OPEN c1 USING :parm_var;
```

- Java™

```
PreparedStatement pstmt1 = con.prepareStatement(
  "SELECT tabname FROM syscat.tables " +
  "WHERE tabname <> ? ORDER BY 1");
```

```
// definir nombre de cursor para la sentencia update posicionada
pstmt1.setCursorName("c1");
pstmt1.setString(1, "STAFF");
ResultSet rs = pstmt1.executeQuery();
```

- COBOL **(dynamic.sqb)**

  El siguiente ejemplo procede del ejemplo **dynamic.sqb**:

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
     01 st              pic x(80).
     01 parm-var        pic x(18).
EXEC SQL END DECLARE SECTION END-EXEC.

move "SELECT TABNAME FROM SYSCAT.TABLES ORDER BY 1 WHERE TABNAME <> ?" to st.
EXEC SQL PREPARE s1 FROM :st END-EXEC.

EXEC SQL DECLARE c1 CURSOR FOR s1 END-EXEC.

move "STAFF" to parm-var.
EXEC SQL OPEN c1 USING :parm-var END-EXEC.
```

EXECUTE IMMEDIATE

También puede preparar y ejecutar una sentencia de SQL dinámico con la sentencia EXECUTE IMMEDIATE (excepto para las sentencias SELECT que devuelven más de una fila).

- C/C++ (**dbuse.sqc/dbuse.sqC**)

  El siguiente ejemplo procede de la función `DynamicStmtEXECUTE_IMMEDIATE()` del ejemplo **dbuse**:

```
EXEC SQL BEGIN DECLARE SECTION;
  char stmt1[50];
EXEC SQL END DECLARE SECTION;

strcpy(stmt1, "CREATE TABLE table1(col1 INTEGER)");
EXEC SQL EXECUTE IMMEDIATE :stmt1;
```

Conceptos relacionados:

- "Recuperación de mensajes de error en una aplicación" en la página 112

Ejemplos relacionados:

- "dbuse.out -- HOW TO USE A DATABASE (C)"
- "dbuse.sqc -- How to use a database (C)"
- "dbuse.out -- HOW TO USE A DATABASE (C++)"
- "dbuse.sqC -- How to use a database (C++)"

Efectos de REOPT en el SQL dinámico

Cuando especifica la opción REOPT ALWAYS, DB2® aplaza la preparación de cualquier sentencia que contenga variables de lenguaje principal, marcadores de parámetros y registros especiales hasta que DB2 encuentre una sentencia OPEN o EXECUTE; es decir, cuando los valores de esas variables pasan a ser conocidos. En ese momento, se genera el plan de acceso utilizando esos valores. Las peticiones OPEN o EXECUTE subsiguientes para la misma sentencia recompilarán la sentencia, reoptimizarán el plan de consulta utilizando el conjunto actual de valores de las variables y ejecutarán el plan de consulta recién creado.

La opción REOPT ONCE tiene un efecto similar, salvo que el plan se optimiza una sola vez utilizando los valores de las variables de lenguaje principal, marcadores de parámetros y de registros especiales. Este plan se coloca en antememoria y es utilizado por peticiones subsiguientes.

Efecto de la opción de vinculación DYNAMICRULES en SQL dinámico

La opción DYNAMICRULES de PRECOMPILE y BIND determina los valores que se aplicarán en el momento de la ejecución para los siguientes atributos de SQL:

- El ID de autorización que se utiliza durante la comprobación de autorización.
- El calificador que se utiliza para la calificación de objetos no calificados.
- Si el paquete se puede utilizar para preparar de forma dinámica las siguientes sentencias: GRANT, REVOKE, ALTER, CREATE, DROP, COMMENT ON, RENAME, SET INTEGRITY y SET EVENT MONITOR STATE.

Además del valor de DYNAMICRULES, el entorno de tiempo de ejecución de un paquete controla el modo en que se comportan las sentencias de SQL en el momento de la ejecución. Los dos posibles entornos de tiempo de ejecución son:

- El paquete se ejecuta formando parte de un programa autónomo
- El paquete se ejecuta dentro del contexto de una rutina

La combinación del valor de DYNAMICRULES y del entorno de tiempo de ejecución determina los valores correspondientes a los atributos de SQL dinámico. Este conjunto de valores de atributos se denomina comportamiento de las sentencias de SQL dinámico. Los cuatro comportamientos son:

Comportamiento de ejecución
DB2® utiliza el ID de autorización del usuario (el ID que se conectó inicialmente a DB2) que ejecuta el paquete como valor que se va a utilizar para la comprobación de autorización de sentencias de SQL dinámico y como valor inicial utilizado para la calificación implícita de referencias a objetos no calificados dentro de sentencias de SQL dinámico.

Comportamiento de vinculación
En el momento de la ejecución, DB2 utiliza todas las normas que se aplican al SQL estático para la autorización y calificación. Es decir, se toma el ID de autorización del propietario del paquete como valor que se va a utilizar para la comprobación de autorización de sentencias de SQL dinámico y el calificador por omisión del paquete para la calificación implícita de referencias a objetos no calificados dentro de sentencias de SQL dinámico.

Comportamiento de definición
El comportamiento de definición sólo se aplica si la sentencia de SQL dinámico está en un paquete que se ejecuta dentro del contexto de una rutina y el paquete se vinculó con DYNAMICRULES DEFINEBIND o DYNAMICRULES DEFINERUN. DB2 utiliza el ID de autorización del definidor de la rutina (no el vinculador de paquetes de la rutina) como valor que se va a utilizar para la comprobación de autorización de sentencias de SQL dinámico y para la calificación implícita de referencias a objetos no calificados dentro de sentencias de SQL dinámico contenidas en dicha rutina.

Comportamiento de invocación

El comportamiento de invocación sólo se aplica si la sentencia de SQL dinámico está en un paquete que se ejecuta dentro del contexto de una rutina y el paquete se vinculó con DYNAMICRULES INVOKEBIND o DYNAMICRULES INVOKERUN. DB2 utiliza el ID de autorización de sentencias actualmente en vigor cuando se invoca la rutina como valor que se va a utilizar para la comprobación de autorización de SQL dinámico y para la calificación implícita de referencias a objetos no calificados dentro de sentencias de SQL dinámico contenidas en dicha rutina. Esto se resume en la tabla siguiente:

Entorno de invocación	**ID utilizado**
Cualquier SQL estático	Valor implícito o explícito del propietario (OWNER) del paquete del que procede el SQL que invoca la rutina.
Utilizado en definición de vista o activador	Definidor de la vista o del activador.
SQL dinámico procedente de un paquete de comportamiento de ejecución	ID utilizado para efectuar la conexión inicial con DB2.
SQL dinámico procedente del paquete de comportamiento de definición	Definidor de la rutina que utiliza el paquete del que procede el SQL que invoca la rutina.
SQL dinámico procedente de un paquete de comportamiento de invocación	ID de autorización Current® que invoca la rutina.

La tabla siguiente muestra la combinación del valor de DYNAMICRULES y el entorno de tiempo de ejecución que da lugar a cada comportamiento del SQL dinámico.

Tabla 10. Cómo DYNAMICRULES y el entorno de tiempo de ejecución determinan el comportamiento de las sentencias de SQL dinámico

Valor de DYNAMICRULES	**Comportamiento de sentencias de SQL dinámico en un entorno de programa autónomo**	**Comportamiento de sentencias de SQL dinámico en un entorno de rutina**
BIND	Comportamiento de vinculación	Comportamiento de vinculación
RUN	Comportamiento de ejecución	Comportamiento de ejecución
DEFINEBIND	Comportamiento de vinculación	Comportamiento de definición
DEFINERUN	Comportamiento de ejecución	Comportamiento de definición
INVOKEBIND	Comportamiento de vinculación	Comportamiento de invocación
INVOKERUN	Comportamiento de ejecución	Comportamiento de invocación

La tabla siguiente muestra los valores de atributos de SQL dinámico correspondientes a cada tipo de comportamiento de SQL dinámico.

Tabla 11. Definiciones de comportamientos de sentencias de SQL dinámico

Atributo de SQL dinámico	Valor correspondiente a atributos de SQL dinámico: comportamiento de vinculación	Valor correspondiente a atributos de SQL dinámico: comportamiento de ejecución	Valor correspondiente a atributos de SQL dinámico: comportamiento de definición	Valor correspondiente a atributos de SQL dinámico: comportamiento de invocación
ID de autorización	Valor implícito o explícito de la opción OWNER BIND	ID de usuario que ejecuta el paquete	Definidor de la rutina (no el propietario del paquete de la rutina)	ID de autorización de sentencias actual cuando se invoca la rutina.
Calificador por omisión para objetos no calificados	Valor implícito o explícito de la opción QUALIFIER BIND	Registro especial CURRENT SCHEMA	Definidor de la rutina (no el propietario del paquete de la rutina)	ID de autorización de sentencias actual cuando se invoca la rutina.
Puede ejecutar GRANT, REVOKE, ALTER, CREATE, DROP, COMMENT ON, RENAME, SET INTEGRITY y SET EVENT MONITOR STATE	No	Sí	No	No

Conceptos relacionados:

- "Consideraciones sobre autorización para SQL dinámico" en la página 51
- "Autorizaciones y vinculación de rutinas que contienen SQL" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*

La SQLDA en programas de SQL dinámico

Las secciones siguientes describen las distintas consideraciones a tener en cuenta cuando se declara la SQLDA para un programa de SQL dinámico.

Variables del lenguaje principal en la SQLDA en programas de SQL dinámico

Con SQL estático, las variables del lenguaje principal utilizadas en sentencias de SQL incorporado se conocen en el momento de compilar la aplicación. Con SQL dinámico, las sentencias de SQL incorporado y por lo tanto las variables del lenguaje principal no se conocen hasta el momento de ejecutar la aplicación. Por lo tanto, para aplicaciones de SQL dinámico, tiene que trabajar con la lista de las variables del lenguaje principal que se utilizan en la aplicación. Puede utilizar la sentencia DESCRIBE para obtener información sobre variables del lenguaje principal para cualquier sentencia SELECT que se haya preparado (mediante PREPARE) y almacenar dicha información en el área del descriptor de SQL (SQLDA).

Nota: Las aplicaciones Java™ no utilizan la estructura SQLDA y, por lo tanto, no utilizan las sentencias PREPARE ni DESCRIBE. En aplicaciones JDBC, puede utilizar un objeto `PreparedStatement` y el método `executeQuery()` para generar un objeto `ResultSet`, que equivale a un cursor del lenguaje

principal. En las aplicaciones SQLJ, también puede declarar un objeto `iterador` de SQLj con un cursor `CursorByPos` o `CursorByName` para devolver datos mediante sentencias FETCH.

Cuando la sentencia DESCRIBE se ejecuta en la aplicación, el gestor de bases de datos define las variables del lenguaje principal en una SQLDA. Cuando las variables del lenguaje principal se han definido en la SQLDA, puede utilizar la sentencia FETCH para asignar valores a las variables del lenguaje principal, mediante un cursor.

Conceptos relacionados:

- "Ejemplo de un cursor en un programa de SQL dinámico" en la página 120

Información relacionada:

- "Sentencia DESCRIBE" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia FETCH" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia PREPARE" en la publicación *Consulta de SQL, Volumen 2*
- "SQLDA" en la publicación *Administrative API Reference*

Declaración de la estructura SQLDA en un programa de SQL dinámico

La SQLDA contiene un número de ocurrencias de variables de entradas SQLVAR, cada una de las cuales contiene un grupo de campos que describen una columna de una fila de datos, tal como se muestra en la siguiente figura. Hay dos tipos de entradas SQLVAR: entradas SQLVAR base y entradas SQLVAR secundarias.

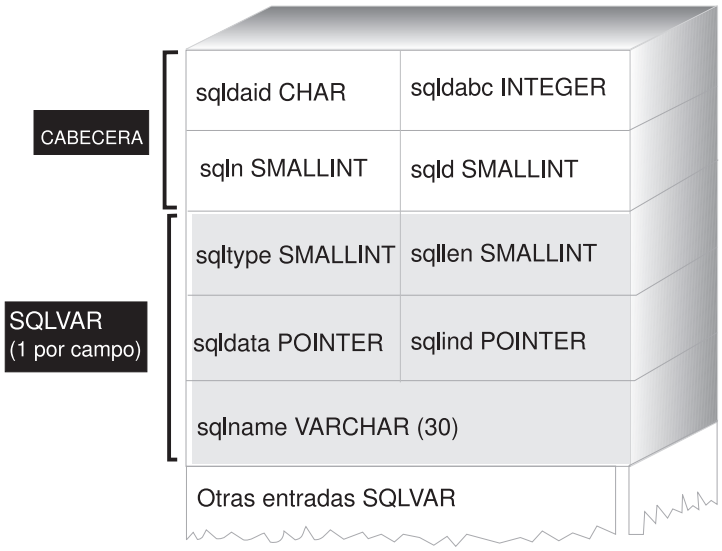


Figura 3. El Área de descriptores de SQL (SQLDA)

Procedimiento:

Puesto que el número de entradas SQLVAR necesario depende del número de columnas de la tabla de resultados, una aplicación debe ser capaz de asignar un número adecuado de elementos SQLVAR cuando se necesiten. Utilice uno de los siguientes métodos:

- Proporcione la SQLDA de mayor tamaño (es decir, la que tenga el mayor número de entradas SQLVAR) que se necesite. El número máximo de columnas que se pueden devolver en una tabla de resultados es 255. Si cualquiera de las columnas devueltas es un tipo LOB o un tipo diferenciado, el valor de SQLN se dobla y el número de entradas SQLVAR necesarias para albergar la información se dobla a 510. Sin embargo, puesto que la mayoría de sentencias SELECT no recuperan ni 255 columnas, la mayoría del espacio asignado no se utiliza.
- Proporcione una SQLDA de menor tamaño con menos entradas SQLVAR. En este caso, si hay más columnas en el resultado que entradas SQLVAR permitidas en la SQLDA, no se devuelve ninguna descripción. En su lugar, el gestor de bases de datos devuelve el número de elementos de lista de selección detectado en la sentencia SELECT. La aplicación asigna una SQLDA con el número de entradas SQLVAR necesario y utiliza la sentencia DESCRIBE para adquirir las descripciones de columnas.

Para ambos métodos, la duda es cuántas entradas SQLVAR iniciales se deben asignar. Cada elemento SQLVAR utiliza un máximo de 44 bytes de almacenamiento (sin contar el almacenamiento asignado para los campos SQLDATA y SQLIND). Si hay memoria suficiente, el primer método de proporcionar una SQLDA de tamaño máximo resulta más fácil de implantar.

El segundo método de asignar una SQLDA de menor tamaño sólo se aplica a lenguajes de programación, como C y C++, que dan soporte a la asignación dinámica de memoria. Para lenguajes como COBOL y FORTRAN, que no dan soporte a la asignación dinámica de memoria, tiene que utilizar el primer método.

Tareas relacionadas:

- "Preparación de una sentencia de SQL dinámico utilizando la estructura SQLDA mínima" en la página 126
- "Asignación de una SQLDA con suficientes entradas SQLVAR para un programa de SQL dinámico" en la página 128
- "Asignación de una estructura SQLDA para un programa de SQL dinámico" en la página 131

Información relacionada:

- "SQLDA" en la publicación *Administrative API Reference*

Preparación de una sentencia de SQL dinámico utilizando la estructura SQLDA mínima

Utilice la información de este tema como ejemplo sobre cómo asignar la estructura SQLDA mínima para una sentencia.

Restricciones:

Sólo puede asignar una estructura SQLDA de menor tamaño con lenguajes de programación, como C y C++, que den soporte a la asignación dinámica de memoria.

Procedimiento:

Supongamos que una aplicación declara una estructura SQLDA denominada `minsqlda` que no contiene ninguna entrada SQLVAR. El campo SQLN de la SQLDA describe el número de entradas SQLVAR que se asignan. En este caso, SQLN se

debe establecer en 0. A continuación, para preparar una sentencia desde la serie de caracteres `dstring` y para entrar su descripción en `minsqlda`, emita la siguiente sentencia de SQL (suponiendo que se utiliza sintaxis de C y que `minsqlda` se declara como un puntero a una estructura SQLDA):

```
EXEC SQL
  PREPARE STMT INTO :*minsqlda FROM :dstring;
```

Supongamos que la sentencia contenida es `dstring` es una sentencia SELECT que devuelve 20 columnas en cada fila. Después de la sentencia PREPARE (o de una sentencia DESCRIBE), el campo SQLD de la SQLDA contiene el número de columnas de la tabla de resultados correspondiente a la sentencia SELECT preparada.

Las entradas SQLVAR de la SQLDA se establecen en los casos siguientes:

- SQLN >= SQLD y ninguna columna tiene el tipo LOB ni un tipo diferenciado.

  Las primeras entradas SQLVAR de SQLD se establecen y SQLDOUBLED se establece en blanco.
- SQLN >= 2*SQLD y al menos una columna es de tipo LOB o tiene un tipo diferenciado.

  2* entradas SQLVAR de SQLD se establecen y SQLDOUBLED se establece en 2.
- SQLD <= SQLN < 2*SQLD y al menos una columna tiene un tipo diferenciado, pero no hay ninguna columna LOB.

  Las primeras entradas SQLVAR de SQLD se establecen y SQLDOUBLED se establece en blanco. Si la opción de vinculación SQLWARN tiene el valor YES, se emite un aviso SQLCODE +237 (SQLSTATE 01594).

Las entradas SQLVAR de la SQLDA *no* se establecen (es necesaria la asignación de espacio adicional y otra sentencia DESCRIBE) en los casos siguientes:

- SQLN < SQLD y ninguna columna tiene el tipo LOB ni un tipo diferenciado.

  No se establece ninguna entrada SQLVAR y SQLDOUBLED se establece en blanco. Si la opción de vinculación SQLWARN tiene el valor YES, se emite un aviso SQLCODE +236 (SQLSTATE 01005).

  Asigne entradas SQLVAR de SQLD para lograr una operación DESCRIBE satisfactoria.
- SQLN < SQLD y al menos una columna tiene un tipo diferenciado, pero no hay ninguna columna LOB.

  No se establece ninguna entrada SQLVAR y SQLDOUBLED se establece en blanco. Si la opción de vinculación SQLWARN tiene el valor YES, se emite un aviso SQLCODE +239 (SQLSTATE 01005).

  Asigne 2*SQLD entradas SQLVAR para lograr una operación DESCRIBE satisfactoria, incluidos los nombres de los tipos diferenciados.
- SQLN < 2*SQLD y al menos una columna es de tipo LOB.

  No se establece ninguna entrada SQLVAR y SQLDOUBLED se establece en blanco. Se emite un aviso SQLCODE +238 (SQLSTATE 01005) (independientemente del valor de la opción de vinculación SQLWARN).

  Asigne 2*SQLD entradas SQLVAR para lograr una operación DESCRIBE satisfactoria.

La opción SQLWARN del mandato BIND sirve para controlar si DESCRIBE (o PREPARE...INTO) devolverá los siguientes avisos:

- SQLCODE +236 (SQLSTATE 01005)
- SQLCODE +237 (SQLSTATE 01594)

- SQLCODE +239 (SQLSTATE 01005).

Se recomienda que el código de la aplicación siempre tenga en cuenta que se pueden devolver estos valores SQLCODE. Siempre se devuelve el aviso SQLCODE +238 (SQLSTATE 01005) cuando hay columnas LOB en la lista de selección y hay un número insuficiente de entradas SQLVAR en la SQLDA. Esta es la única manera que tiene la aplicación para saber que el número de entradas SQLVAR se debe doblar debido a una columna LOB en el conjunto de resultados.

Tareas relacionadas:

- "Declaración de la estructura SQLDA en un programa de SQL dinámico" en la página 125
- "Asignación de una SQLDA con suficientes entradas SQLVAR para un programa de SQL dinámico" en la página 128
- "Asignación de una estructura SQLDA para un programa de SQL dinámico" en la página 131

Asignación de una SQLDA con suficientes entradas SQLVAR para un programa de SQL dinámico

Después de determinar el número de columnas de la tabla de resultados, asigne almacenamiento para un segundo SQLDA de tamaño completo.

Procedimiento:

Supongamos que la tabla de resultados tiene 20 columnas (ninguna de las cuales es de tipo LOB). En esta situación, debe asignar una segunda estructura SQLDA, `fulsqlda`, con un mínimo de 20 elementos SQLVAR (o 40 elementos si la tabla de resultados contiene algún LOB o tipo diferenciado). Para el resto de este ejemplo, supongamos que no hay ningún LOB ni tipo diferenciado en la tabla de resultados.

Cuando calcule los requisitos de almacenamiento para estructuras de SQLDA, incluya lo siguiente:

- Una cabecera de longitud fija, de 16 bytes de longitud, que contiene campos como SQLN y SQLD
- Una matriz de longitud variable de entradas SQLVAR, de la cual cada elemento tiene 44 bytes de longitud en plataformas de 32 bits y 56 bytes de longitud en plataformas de 64 bits.

El número de entradas SQLVAR necesarias para `fulsqlda` se especifica en el campo SQLD de `minsqlda`. Supongamos que este valor es 20. Por lo tanto, la asignación de almacenamiento necesaria para `fulsqlda` es:

```
16 + (20 * sizeof(struct sqlvar))
```

Nota: En plataformas de 64 bits, `sizeof(struct sqlvar)` y `sizeof(struct sqlvar2)` devuelven `56`. En plataformas de 32 bits, `sizeof(struct sqlvar)` y `sizeof(struct sqlvar2)` devuelven `44`.

Este valor representa el tamaño de la cabecera más 20 multiplicado por el tamaño de cada entrada SQLVAR, lo que da un total de 896 bytes.

Puede utilizar la macro SQLDASIZE para no tener que realizar sus propios cálculos y para evitar dependencias específicas de cada versión.

Tareas relacionadas:

- "Declaración de la estructura SQLDA en un programa de SQL dinámico" en la página 125
- "Preparación de una sentencia de SQL dinámico utilizando la estructura SQLDA mínima" en la página 126
- "Asignación de una estructura SQLDA para un programa de SQL dinámico" en la página 131

Descripción de una sentencia SELECT en un programa de SQL dinámico

Después de asignar espacio suficiente para el segundo SQLDA (en este ejemplo, denominado `fulsqlda`), debe codificar la aplicación para que describa la sentencia SELECT.

Procedimiento:

Codifique la aplicación para que lleve a cabo los pasos siguientes:

1. Almacenar el valor `20` en el campo SQLN de `fulsqlda` (en este ejemplo se supone que la tabla de resultados contiene 20 columnas y que ninguna de ellas es una columna LOB).
2. Obtener información sobre la sentencia SELECT mediante la segunda estructura SQLDA, `fulsqlda`. Hay dos métodos disponibles:
   - Utilizar otra sentencia PREPARE, especificando `fulsqlda` en lugar de `minsqlda`.
   - Utilizar la sentencia DESCRIBE especificando `fulsqlda`.

Es preferible utilizar la sentencia DESCRIBE porque se evita el coste de preparar la sentencia por segunda vez. La sentencia DESCRIBE simplemente reutiliza la información obtenida previamente durante la operación de preparación para llenar la nueva estructura SQLDA. Se puede emitir la siguiente sentencia:

```
EXEC SQL DESCRIBE STMT INTO :fulsqlda
```

Una vez ejecutada esta sentencia, cada elemento SQLVAR contiene una descripción de una columna de la tabla de resultados.

Tareas relacionadas:

- "Adquisición de almacenamiento para albergar una fila" en la página 129

Adquisición de almacenamiento para albergar una fila

Para que una aplicación pueda captar una fila de la tabla de resultados utilizando una estructura SQLDA, la aplicación debe antes asignar almacenamiento para la fila.

Procedimiento:

Codifique la aplicación de modo que haga lo siguiente:

1. Analice cada descripción de SQLVAR para determinar cuánto espacio se necesita para el valor de dicha columna.

   Tenga en cuenta que, para valores LOB, cuando se describe la sentencia SELECT, los datos que se proporcionan en la SQLVAR son SQL_TYP_*x*LOB.

Este tipo de datos corresponde a una variable del lenguaje principal LOB plana, es decir, el LOB completo se almacena en memoria de una sola vez. Esto funciona para LOB pequeños (de un máximo de unos pocos MB), pero no puede utilizar este tipo de datos para LOB grandes (por ejemplo, de 1 GB). Será necesario que la aplicación cambie su definición de columna en la SQLVAR para que sea SQL_TYP_*x*LOB_LOCATOR o SQL_TYPE_*x*LOB_FILE. (Tenga en que cuenta que, si se cambia el campo SQLTYPE de la SQLVAR, también se tiene que cambiar el campo SQLLEN.) Después de cambiar la definición de columna en la SQLVAR, la aplicación puede asignar la cantidad correcta de almacenamiento correspondiente al nuevo tipo.

2. Asigne almacenamiento para el valor de dicha columna.
3. Almacene la dirección del almacenamiento asignado en el campo SQLDATA de la estructura SQLDA.

Estos pasos se deben llevar a cabo analizando la descripción de cada columna y sustituyendo el contenido de cada campo SQLDATA por la dirección de un área de almacenamiento lo suficientemente grande como para albergar cualquier valor procedente de dicha columna. El atributo de longitud se determina a partir del campo SQLLEN de cada entrada SQLVAR correspondiente a elementos de datos que no son de tipo LOB. Para elementos con un tipo BLOB, CLOB o DBCLOB, el atributo de longitud se determina a partir del campo SQLLONGLEN de la entrada SQLVAR secundaria.

Además, si la columna especificada permite nulos, la aplicación debe sustituir el contenido del campo SQLIND por la dirección de una variable de indicador correspondiente a la columna.

Conceptos relacionados:

- "Uso de objetos grandes" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*

Tareas relacionadas:

- "Proceso del cursor en un programa de SQL dinámico" en la página 130

Proceso del cursor en un programa de SQL dinámico

Después de asignar correctamente la estructura SQLDA, el cursor asociado con la sentencia SELECT se puede abrir y se pueden captar filas.

Procedimiento:

Para procesar el cursor asociado con una sentencia SELECT, primero abra el cursor y luego capte filas especificando la cláusula USING DESCRIPTOR de la sentencia FETCH. Por ejemplo, una aplicación C podría tener lo siguiente:

```
EXEC SQL OPEN pcurs
EMB_SQL_CHECK( "OPEN" ) ;
EXEC SQL FETCH pcurs USING DESCRIPTOR :*sqldaPointer
EMB_SQL_CHECK( "FETCH" ) ;
```

Para procesar una operación FETCH satisfactoriamente, podría escribir la aplicación de modo que obtuviera los datos de la SQLDA y mostrara las cabeceras de columna. Por ejemplo:

```
display_col_titles( sqldaPointer ) ;
```

Una vez visualizados los datos, debería cerrar el cursor y liberar la memoria asignada de forma dinámica. Por ejemplo:

```
EXEC SQL CLOSE pcurs ;
EMB_SQL_CHECK( "CLOSE CURSOR" ) ;
```

Asignación de una estructura SQLDA para un programa de SQL dinámico

Asigne una estructura SQLDA correspondiente a la aplicación de modo que pueda utilizarla para pasar datos a la aplicación y para recibir datos de esta.

Procedimiento:

Para crear una estructura SQLDA con C, incorpore la sentencia INCLUDE SQLDA en el lenguaje principal o incluya el archivo include de SQLDA para obtener la definición de estructura. Luego, debido a que el tamaño de una SQLDA no es fijo, la aplicación debe declarar un puntero a una estructura SQLDA y asignar almacenamiento para la misma. El tamaño real de la estructura SQLDA depende del número de elementos de datos diferenciados que se pasan mediante la SQLDA.

En el lenguaje de programación C/C++, se proporciona una macro que facilita la asignación de SQLDA. Con la excepción de la plataforma HP-UX, esta macro tiene el siguiente formato:

```
#define SQLDASIZE(n) (offsetof(struct sqlda, sqlvar) \
        + (n) * sizeof(struct sqlvar))
```

En la plataforma HP-UX, la macro tiene el siguiente formato:

```
#define SQLDASIZE(n) (sizeof(struct sqlda) \
        + (n-1) * sizeof(struct sqlvar))
```

El efecto de esta macro es calcular el almacenamiento necesario para una SQLDA con `n` elementos SQLVAR.

Para crear una estructura SQLDA con COBOL, puede incorporar una sentencia INCLUDE SQLDA o utilizar la sentencia COPY. Utilice la sentencia COPY cuando desee controlar el número máximo de entradas SQLVAR y por tanto la cantidad de espacio de almacenamiento ocupado por la SQLDA. Por ejemplo, para cambiar el número por omisión de entradas SQLVAR de 1489 a 1, utilice la siguiente sentencia COPY:

```
COPY "sqlda.cbl"
       replacing --1489--
       by --1--.
```

El lenguaje FORTRAN no da soporte directamente a las estructuras de datos autodefinidas ni a la asignación dinámica. No se proporciona ningún archivo include de SQLDA para FORTRAN porque no es posible dar soporte al SQLDA como una estructura de datos en FORTRAN. El precompilador pasará por alto la sentencia INCLUDE SQLDA en un programa FORTRAN.

Sin embargo, puede crear algo parecido a una estructura SQLDA estática en un programa FORTRAN y utilizar dicha estructura siempre que se pueda utilizar una SQLDA. El archivo `sqldact.f` contiene constantes que ayudan a declarar una estructura SQLDA en FORTRAN.

Ejecute llamadas a SQLGADDR para asignar valores de puntero a elementos de SQLDA que los necesiten.

La tabla siguiente muestra la declaración y uso de una estructura SQLDA con un elemento SQLVAR.

Lenguaje **Código fuente de ejemplo**

C/C++

```
#include <sqlda.h>
struct sqlda *outda = (struct sqlda *)malloc(SQLDASIZE(1));

/* DECLARAR VARIABLES LOCALES PARA ALBERGAR DATOS REALES */
double sal;
double sal = 0;
short salind;
short salind = 0;

/* INICIALIZAR UN ELEMENTO DE SQLDA */
memcpy( outda->sqldaid,"SQLDA   ",sizeof(outda->sqldaid));
outda->sqln = outda->sqld = 1;
outda->sqlvar[0].sqltype = SQL_TYP_NFLOAT;
outda->sqlvar[0].sqllen  = sizeof( double );.
outda->sqlvar[0].sqldata = (unsigned char *)&sal;
outda->sqlvar[0].sqlind  = (short *)&salind;
```

COBOL

```
       WORKING-STORAGE SECTION.
       77 SALARY          PIC S99999V99 COMP-3.
       77 SAL-IND         PIC S9(4)     COMP-5.

          EXEC SQL INCLUDE SQLDA END-EXEC

      * O codificar un modo útil para guardar entradas SQLVAR no utilizadas.
      * COPY "sqlda.cbl" REPLACING --1489-- BY --1--.

            01 decimal-sqllen pic s9(4) comp-5.
            01 decimal-parts redefines decimal-sqllen.
               05 precision pic x.
               05 scale pic x.

      * Inicializar un elemento de SQLDA de salida
            MOVE 1 TO SQLN
            MOVE 1 TO SQLD
            MOVE SQL-TYP-NDECIMAL TO SQLTYPE(1)

      * Longitud = 7 dígitos de precisión y 2 dígitos de escala

            MOVE x"07" TO PRECISION.
            MOVE x"02" TO SCALE.
            MOVE DECIMAL-SQLLEN TO O-SQLLEN(1).
            SET SQLDATA(1) TO ADDRESS OF SALARY
            SET SQLIND(1)  TO ADDRESS OF SAL-IND
```

Lenguaje	Código fuente de ejemplo

FORTRAN

```
      include 'sqldact.f'

      integer*2  sqlvar1
      parameter ( sqlvar1 = sqlda_header_sz + 0*sqlvar_struct_sz )

C     Declarar una SQLDA de salida -- 1 Variable
      character    out_sqlda(sqlda_header_sz + 1*sqlvar_struct_sz)

      character*8  out_sqldaid     ! Cabecera
      integer*4    out_sqldabc
      integer*2    out_sqln
      integer*2    out_sqld

      integer*2    out_sqltype1    ! Primera variable
      integer*2    out_sqllen1
      integer*4    out_sqldata1
      integer*4    out_sqlind1
      integer*2    out_sqlname1
      character*30 out_sqlnamec1

      equivalence( out_sqlda(sqlda_sqldaid_ofs), out_sqldaid )
      equivalence( out_sqlda(sqlda_sqldabc_ofs), out_sqldabc )
      equivalence( out_sqlda(sqlda_sqln_ofs), out_sqln       )
      equivalence( out_sqlda(sqlda_sqld_ofs), out_sqld       )
      equivalence( out_sqlda(sqlvar1+sqlvar_type_ofs), out_sqltype1 )
      equivalence( out_sqlda(sqlvar1+sqlvar_len_ofs), out_sqllen1   )
      equivalence( out_sqlda(sqlvar1+sqlvar_data_ofs), out_sqldata1 )
      equivalence( out_sqlda(sqlvar1+sqlvar_ind_ofs), out_sqlind1   )
      equivalence( out_sqlda(sqlvar1+sqlvar_name_length_ofs),
     +             out_sqlname1                                    )
      equivalence( out_sqlda(sqlvar1+sqlvar_name_data_ofs),
     +             out_sqlnamec1                                   )

C     Declarar variables locales para albergar los datos devueltos.
      real*8       salary
      integer*2    sal_ind

C     Inicializar la SQLDA de salida (cabecera)
      out_sqldaid  = 'OUT_SQLDA'
      out_sqldabc  = sqlda_header_sz + 1*sqlvar_struct_sz
      out_sqln     = 1
      out_sqld     = 1
C     Inicializar VAR1
      out_sqltype1 = SQL_TYP_NFLOAT
      out_sqllen1  = 8
      rc = sqlgaddr( %ref(salary), %ref(out_sqldata1) )
      rc = sqlgaddr( %ref(sal_ind), %ref(out_sqlind1) )
```

En lenguajes que no dan soporte a la asignación dinámica de memoria, se debe declarar de forma explícita en el lenguaje principal una SQLDA con el número deseado de elementos SQLVAR. Asegúrese de declarar el número suficiente de elementos SQLVAR según los requisitos de la aplicación.

Tareas relacionadas:

- "Preparación de una sentencia de SQL dinámico utilizando la estructura SQLDA mínima" en la página 126
- "Asignación de una SQLDA con suficientes entradas SQLVAR para un programa de SQL dinámico" en la página 128
- "Transferencia de datos en un programa de SQL dinámico mediante una estructura SQLDA" en la página 134

Transferencia de datos en un programa de SQL dinámico mediante una estructura SQLDA

Puede obtener una mayor flexibilidad si transfiere datos mediante una SQLDA que si utiliza listas de variables del lenguaje principal. Por ejemplo, puede utilizar una SQLDA para transferir datos que no tienen ningún equivalente en el lenguaje principal, como datos DECIMAL en lenguaje C.

Procedimiento:

Utilice la siguiente tabla como listado de referencias cruzadas que muestra cómo se relacionan los valores numéricos y los nombres simbólicos.

Tabla 12. Tipos de SQL de SQLDA de DB2. Valores numéricos y nombres simbólicos correspondientes

Tipo de columna SQL	Valor numérico SQLTYPE	Nombre simbólico SQLTYPE[1]
DATE	384/385	SQL_TYP_DATE / SQL_TYP_NDATE
TIME	388/389	SQL_TYP_TIME / SQL_TYP_NTIME
TIMESTAMP	392/393	SQL_TYP_STAMP / SQL_TYP_NSTAMP
n/d[2]	400/401	SQL_TYP_CGSTR / SQL_TYP_NCGSTR
BLOB	404/405	SQL_TYP_BLOB / SQL_TYP_NBLOB
CLOB	408/409	SQL_TYP_CLOB / SQL_TYP_NCLOB
DBCLOB	412/413	SQL_TYP_DBCLOB / SQL_TYP_NDBCLOB
VARCHAR	448/449	SQL_TYP_VARCHAR / SQL_TYP_NVARCHAR
CHAR	452/453	SQL_TYP_CHAR / SQL_TYP_NCHAR
LONG VARCHAR	456/457	SQL_TYP_LONG / SQL_TYP_NLONG
n/d[3]	460/461	SQL_TYP_CSTR / SQL_TYP_NCSTR
VARGRAPHIC	464/465	SQL_TYP_VARGRAPH / SQL_TYP_NVARGRAPH
GRAPHIC	468/469	SQL_TYP_GRAPHIC / SQL_TYP_NGRAPHIC
LONG VARGRAPHIC	472/473	SQL_TYP_LONGRAPH / SQL_TYP_NLONGRAPH
FLOAT	480/481	SQL_TYP_FLOAT / SQL_TYP_NFLOAT
REAL[4]	480/481	SQL_TYP_FLOAT / SQL_TYP_NFLOAT
DECIMAL[5]	484/485	SQL_TYP_DECIMAL / SQL_TYP_DECIMAL
INTEGER	496/497	SQL_TYP_INTEGER / SQL_TYP_NINTEGER
SMALLINT	500/501	SQL_TYP_SMALL / SQL_TYP_NSMALL
n/d	804/805	SQL_TYP_BLOB_FILE / SQL_TYPE_NBLOB_FILE
n/d	808/809	SQL_TYP_CLOB_FILE / SQL_TYPE_NCLOB_FILE
n/d	812/813	SQL_TYP_DBCLOB_FILE / SQL_TYPE_NDBCLOB_FILE
n/d	960/961	SQL_TYP_BLOB_LOCATOR / SQL_TYP_NBLOB_LOCATOR
n/d	964/965	SQL_TYP_CLOB_LOCATOR / SQL_TYP_NCLOB_LOCATOR
n/d	968/969	SQL_TYP_DBCLOB_LOCATOR / SQL_TYP_NDBCLOB_LOCATOR

Tabla 12. Tipos de SQL de SQLDA de DB2 (continuación). Valores numéricos y nombres simbólicos correspondientes

Tipo de columna SQL	Valor numérico SQLTYPE	Nombre simbólico SQLTYPE[1]

Nota: Estos tipos definidos se encuentran en el archivo include `sql.h` del subdirectorio `include` del directorio `sqllib`. (Por ejemplo, `sqllib/include/sql.h` para el lenguaje de programación C.)

1. Para el lenguaje de programación COBOL, el nombre SQLTYPE no utiliza el signo de subrayado (_) sino un guión (-).
2. Es una serie gráfica terminada en nulo.
3. Es una serie de caracteres terminada en nulo.
4. La diferencia entre REAL y DOUBLE en la SQLDA es el valor de la longitud (4 u 8).
5. La precisión está en el primer byte. La escala está en el segundo byte.

Tareas relacionadas:

- "Descripción de una sentencia SELECT en un programa de SQL dinámico" en la página 129
- "Adquisición de almacenamiento para albergar una fila" en la página 129
- "Proceso del cursor en un programa de SQL dinámico" en la página 130

Proceso de sentencias interactivas de SQL en programas de SQL dinámico

Una aplicación que utiliza SQL dinámico se puede escribir de modo que procese sentencias arbitrarias de SQL. Por ejemplo, si una aplicación acepta sentencias de SQL del usuario, la aplicación debe ser capaz de ejecutar las sentencias sin conocimiento previo de las mismas.

Procedimiento:

Utilice las sentencias PREPARE y DESCRIBE con una estructura SQLDA de modo que la aplicación pueda determinar el tipo de sentencia de SQL que se ejecutan y actuar consecuentemente.

Conceptos relacionados:

- "Determinación del tipo se sentencia en programas de SQL dinámico" en la página 135

Determinación del tipo se sentencia en programas de SQL dinámico

Cuando se prepara una sentencia de SQL, la información sobre el tipo de sentencia se puede determinar examinando la estructura SQLDA. Esta información se coloca en la estructura SQLDA en el momento de la preparación de la sentencia con la cláusula INTO o emitiendo una sentencia DESCRIBE contra una sentencia preparada anteriormente.

En cualquier caso, el gestor de bases de datos coloca un valor en el campo SQLD de la estructura SQLDA que indica el número de columnas de la tabla de resultados generada por la sentencia de SQL. Si el campo SQLD contiene un cero (0), significa que la sentencia *no* es una sentencia SELECT. Puesto que la sentencia ya está preparada, se puede ejecutar inmediatamente mediante la sentencia EXECUTE.

Si la sentencia contiene marcadores de parámetros, se debe especificar la cláusula USING. La cláusula USING puede especificar una lista de variables del lenguaje principal o una estructura SQLDA.

Si el campo SQLD es mayor que cero, significa que la sentencia es una sentencia SELECT y se debe procesar tal como se describe en las siguientes secciones.

Información relacionada:

- "Sentencia EXECUTE" en la publicación *Consulta de SQL, Volumen 2*

Proceso de sentencias SELECT de lista de variables en programas de SQL dinámico

Una sentencia SELECT de *lista de variables* es una sentencia en la que el número y los tipos de columnas que se tienen que devolver no se conocen en el momento de la precompilación. En este caso, la aplicación no sabe con antelación las variables exactas del lenguaje principal que se tienen que declarar para albergar una fila de la tabla de resultados.

Procedimiento:

Para procesar una sentencia SELECT de lista de variables, codifique la aplicación de modo que haga lo siguiente:

1. Declare una SQLDA.

   Se debe utilizar una estructura SQLDA para procesar las sentencias SELECT de lista de variables.
2. PREPARE la sentencia mediante la cláusula INTO.

   Luego la aplicación determina si la estructura SQLDA declarada tiene suficientes elementos SQLVAR. Si no es así, la aplicación asigna otra estructura SQLDA con el número necesario de elementos SQLVAR y emite una sentencia DESCRIBE adicional mediante el nuevo SQLDA.
3. Asigne los elementos SQLVAR.

   Asigne almacenamiento para las variables del lenguaje principal e indicadores necesarios para cada SQLVAR. Este paso incluye la colocación de las direcciones asignadas para los datos y variables de indicador en cada elemento SQLVAR.
4. Procese la sentencia SELECT.

   Se asocia un cursor con la sentencia preparada, se abre y las filas se captan mediante la estructura SQLDA correctamente asignada.

Tareas relacionadas:

- "Declaración de la estructura SQLDA en un programa de SQL dinámico" en la página 125
- "Preparación de una sentencia de SQL dinámico utilizando la estructura SQLDA mínima" en la página 126
- "Asignación de una SQLDA con suficientes entradas SQLVAR para un programa de SQL dinámico" en la página 128
- "Descripción de una sentencia SELECT en un programa de SQL dinámico" en la página 129
- "Adquisición de almacenamiento para albergar una fila" en la página 129
- "Proceso del cursor en un programa de SQL dinámico" en la página 130

Cómo guardar peticiones de SQL procedentes de usuarios finales

Si los usuarios de la aplicación pueden emitir peticiones de SQL desde la aplicación, es posible que desee guardar dichas peticiones.

Procedimiento:

Si la aplicación permite a los usuarios guardar sentencias arbitrarias de SQL, las puede guardar en una tabla con una columna que tenga el tipo VARCHAR, LONG VARCHAR, CLOB, VARGRAPHIC, LONG VARGRAPHIC o DBCLOB. Tenga en cuenta que los tipos de datos VARGRAPHIC, LONG VARGRAPHIC y DBCLOB sólo están disponibles en entornos de juego de caracteres de doble byte (DBCS) y de Extended UNIX Code (EUC).

Debe guardar los elementos SQL fuente, no las versiones preparadas. Esto significa que debe recuperar y luego preparar cada sentencia antes de ejecutar la versión almacenada en la tabla. Básicamente, la aplicación prepara una sentencia de SQL a partir de una serie de caracteres y ejecuta esta sentencia de forma dinámica.

Marcadores de parámetros en programas de SQL dinámico

Las secciones siguientes describen cómo utilizar marcadores de parámetros para proporcionar entrada de variables a un programa de SQL dinámico y describen brevemente los programas de ejemplo que utilizan marcadores de parámetros.

Cómo proporcionar entrada de variables a SQL dinámico mediante marcadores de parámetros

Una sentencia de SQL dinámico no puede contener variables del lenguaje principal, porque la información de las variables del lenguaje principal (tipo de datos y longitud) sólo está disponible durante la precompilación de la aplicación. En el momento de la ejecución, la información de las variables del lenguaje principal no está disponible.

En SQL dinámico, se utilizan marcadores de parámetros en lugar de variables del lenguaje principal. Los marcadores de parámetros se indican mediante un signo de interrogación (?) e indican dónde se debe sustituir una variable del lenguaje principal dentro de una sentencia de SQL.

Procedimiento:

Supongamos que la aplicación utiliza SQL dinámico y que desea que pueda realizar una operación DELETE. Una serie de caracteres que contenga un marcador de parámetros puede tener un aspecto parecido al siguiente:

```
DELETE FROM TEMPL WHERE EMPNO = ?
```

Cuando se ejecuta esta sentencia, se especifica una variable del lenguaje principal o una estructura SQLDA mediante la cláusula USING de la sentencia EXECUTE. El contenido de la variable del lenguaje principal se utiliza cuando se ejecuta la sentencia.

El marcador de parámetros adopta un tipo de datos y longitud supuestos que dependen del contexto de su uso dentro de la sentencia de SQL. Si el tipo de datos de un marcador de parámetros no resulta obvio a partir del contexto de la

sentencia en la que se utiliza, utilice CAST para especificar el tipo. Dicho marcador de parámetros se considera un a *marcador de parámetros tipificado*. Los marcadores de parámetros tipificados se tratan como una variable del lenguaje principal del tipo determinado. Por ejemplo, la sentencia `SELECT ? FROM SYSCAT.TABLES` no es válida porque DB2 no conoce el tipo de la columna de resultados. Sin embargo, la sentencia `SELECT CAST(? AS INTEGER) FROM SYSCAT.TABLES` es válida porque cast indica que el marcador de parámetros representa un INTEGER, de modo que DB2 conoce el tipo de la columna de resultados.

Si la sentencia de SQL contiene más de un marcador de parámetros, la cláusula USING de la sentencia EXECUTE debe especificar una lista de variables del lenguaje principal (una para cada marcador de parámetros) o debe identificar una SQLDA que tenga una entrada SQLVAR para cada marcador de parámetros. (Observe que para los LOB, existen dos entradas SQLVAR por cada marcador de parámetros.) La lista de variables del lenguaje principal o entradas SQLVAR se comparan de acuerdo con el orden de los marcadores de parámetros en la sentencia y deben tener tipos de datos compatibles.

Nota: El utilizar un marcador de parámetros con SQL dinámico es como utilizar variables del lenguaje principal con SQL estático. En cualquier caso, el optimizador no utiliza estadísticas de distribución y es posible que no elija el mejor plan de acceso.

Las normas que se aplican a marcadores de parámetros se describen con la sentencia PREPARE.

Información relacionada:

- "Sentencia PREPARE" en la publicación *Consulta de SQL, Volumen 2*

Ejemplo de marcadores de parámetros en un programa de SQL dinámico

Los siguientes ejemplos muestran cómo utilizar marcadores de parámetros en un programa de SQL dinámico:

- C/C++ (**dbuse.sqc/dbuse.sqC**)

  La función `DynamicStmtWithMarkersEXECUTEusingHostVars()` del ejemplo en lenguaje C **dbuse.sqc** muestra cómo realizar una supresión mediante un marcador de parámetros con una variable del lenguaje principal:

```
EXEC SQL BEGIN DECLARE SECTION;
  char hostVarStmt1[50];
  short hostVarDeptnumb;
EXEC SQL END DECLARE SECTION;

/* preparar la sentencia con un marcador de parámetros */
strcpy(hostVarStmt1, "DELETE FROM org WHERE deptnumb = ?");
EXEC SQL PREPARE Stmt1 FROM :hostVarStmt1;

/* ejecutar la sentencia correspondiente a hostVarDeptnumb = 15 */
hostVarDeptnumb = 15;
EXEC SQL EXECUTE Stmt1 USING :hostVarDeptnumb;
```

- JDBC (**DbUse.java**)

  La función `execPreparedStatementWithParam()` del ejemplo JDBC **DbUse.java** muestra cómo realizar una supresión mediante marcadores de parámetros:

```
// preparar la sentencia con marcadores de parámetros
PreparedStatement prepStmt = con.prepareStatement(
  " DELETE FROM org WHERE deptnumb <= ? AND division = ? ");

// ejecutar la sentencia
prepStmt.setInt(1, 70);
prepStmt.setString(2, "Eastern");
prepStmt.execute();

// cerrar la sentencia
prepStmt.close();
```

- COBOL (**varinp.sqb**)

  El siguiente ejemplo procede del ejemplo de COBOL **varinp.sqb** y muestra cómo utilizar un marcador de parámetros en condiciones de búsqueda y actualización:

```
   EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 pname             pic x(10).
01 dept              pic s9(4) comp-5.
01 st                pic x(127).
01 parm-var          pic x(5).
   EXEC SQL END DECLARE SECTION END-EXEC.

   move "SELECT name, dept FROM staff
-         "   WHERE job = ? FOR UPDATE OF job" to st.
   EXEC SQL PREPARE s1 FROM :st END-EXEC.

   EXEC SQL DECLARE c1 CURSOR FOR s1 END-EXEC.

   move "Mgr" to parm-var.
   EXEC SQL OPEN c1 USING :parm-var END-EXEC

   move "Clerk" to parm-var.
   move "UPDATE staff SET job = ? WHERE CURRENT OF c1" to st.
   EXEC SQL PREPARE s2 from :st END-EXEC.

* llamar a FETCH y al bucle UPDATE.
   perform Fetch-Loop thru End-Fetch-Loop
      until SQLCODE not equal 0.

   EXEC SQL CLOSE c1 END-EXEC.
```

Conceptos relacionados:

- "Recuperación de mensajes de error en una aplicación" en la página 112

Ejemplos relacionados:

- "dbuse.out -- HOW TO USE A DATABASE (C)"
- "dbuse.sqc -- How to use a database (C)"
- "dbuse.out -- HOW TO USE A DATABASE (C++)"
- "dbuse.sqC -- How to use a database (C++)"
- "DbUse.java -- How to use a database (JDBC)"
- "DbUse.out -- HOW TO USE A DATABASE. Connect to 'sample' database using JDBC type 2 driver (JDBC)"

Comparación entre Interfaz de nivel de llamada (CLI) de DB2 y SQL dinámico

Las secciones siguientes describen las diferencias entre CLI de DB2 y SQL dinámico, las ventajas de CLI de DB2 sobre SQL dinámico y cuándo debe utilizar CLI de DB2 o SQL dinámico.

La Interfaz de nivel de llamada (CLI) de DB2 y el SQL dinámico incorporado

Una aplicación que utiliza una interfaz de SQL incorporado necesita un precompilador para convertir las sentencias de SQL en código, que luego se puede compilar, vincular a la base de datos y ejecutar. Por el contrario, una aplicación CLI de DB2 no se tiene que precompilar ni vincular, pero utiliza un conjunto estándar de funciones para ejecutar sentencias de SQL y servicios relacionados en el tiempo de ejecución.

Esta diferencia es importante porque, tradicionalmente, los precompiladores han sido específicos para cada producto de base de datos, lo que establece una relación entre las aplicaciones y dicho producto. CLI de DB2 le permite escribir aplicaciones portables que no dependen de un determinado producto de base de datos. Esta independencia significa que las aplicaciones CLI de DB2 no se tienen que recompilar y revincular para poder acceder a distintas bases de datos de DB2®, incluidas las bases de datos del sistema principal. Simplemente se conectan a la base de datos adecuada en el tiempo de ejecución.

A continuación se muestran las diferencias y similitudes entre CLI de DB2 y SQL incorporado:

- CLI de DB2 no necesita la declaración explícita de cursores. CLI de DB2 tiene un suministro de cursores que se utilizan cuando hace falta. Luego la aplicación puede utilizar el cursor generado en el modelo de captación de cursor normal para varias sentencias `SELECT` de fila y sentencias `UPDATE` y `DELETE` colocadas.
- La sentencia `OPEN` no se utiliza en CLI de DB2. En su lugar, la ejecución de `SELECT` hace que se abra un cursor de inmediato.
- A diferencia de SQL incorporado, CLI de DB2 permite el uso de marcadores de parámetros en el equivalente de la sentencia `EXECUTE IMMEDIATE` (la función `SQLExecDirect()`).
- Una sentencia `COMMIT` o `ROLLBACK` en CLI de DB2 se suele emitir mediante la función `SQLEndTran()` en lugar de ejecutarse como una sentencia de SQL, aunque esto último está permitido.
- CLI de DB2 gestiona la información relacionada con sentencias en nombre de la aplicación y proporciona un objeto abstracto para representar la información que se denomina *descriptor de contexto de sentencias*. Este descriptor de contexto elimina la necesidad de que la aplicación utilice estructuras de datos específicas del producto.
- Parecidos al descriptor de contexto de sentencias, el *descriptor de contexto de entorno* y el *descriptor de contexto de conexiones* proporcionan métodos para hacer referencia a variables globales y a información específica de la conexión. El *descriptor de contexto descriptor* describe los parámetros de una sentencia de SQL o las columnas de un conjunto de resultados.
- Las aplicaciones CLI de DB2 pueden describir parámetros de forma dinámica en una sentencia de SQL del mismo modo que las aplicaciones CLI y de SQL incorporado describen los conjuntos de resultados. Esto permite a las

aplicaciones CLI procesar de forma dinámica sentencias de SQL que contienen marcadores de parámetros sin saber de forma anticipada el tipo de datos de dichos marcadores de parámetros. Cuando se prepara la sentencia de SQL, se devuelve información de descripción que detalla los tipos de datos de los parámetros.

- CLI de DB2 utiliza los valores de SQLSTATE definidos por la especificación CAE de SQL de X/Open. Aunque el formato y la mayoría de los valores son coherentes con los valores que utilizan los productos de bases de datos relacionales de IBM®, hay algunas diferencias. (También hay diferencias entre los SQLSTATES de ODBC y los SQLSTATES definidos por X/Open).

A pesar de estas diferencias, hay un importante concepto común entre SQL incorporado y CLI de DB2: *CLI de DB2 puede ejecutar cualquier sentencia de SQL que se puede preparar de forma dinámica en SQL incorporado.*

Nota: CLI de DB2 también puede aceptar algunas sentencias de SQL que no se pueden preparar de forma dinámica, como sentencias compuestas de SQL.

Cada DBMS puede tener sentencias adicionales que el usuario puede preparar de forma dinámica. En este caso, CLI de DB2 pasa las sentencias directamente al DBMS. Hay una excepción: algunos DBMS pueden preparar las sentencias COMMIT y ROLLBACK de forma dinámica, pero CLI de DB2 las interceptará y las tratará como una petición `SQLEndTran()` adecuada. Sin embargo, se recomienda utilizar la función `SQLEndTran()` para especificar la sentencia COMMIT o ROLLBACK.

Información relacionada:

- Apéndice A, “Sentencias de SQL soportadas”, en la página 733

Ventajas de CLI de DB2 sobre el SQL incorporado

La interfaz CLI de DB2 tiene varias ventajas clave sobre SQL incorporado.

- Resulta ideal para un entorno cliente-servidor, en el que la base de datos de destino no se conoce cuando se crea la aplicación. Proporciona una interfaz coherente para ejecutar sentencias de SQL, independientemente del servidor de bases de datos al que se conecte la aplicación.
- Aumenta la portabilidad de las aplicaciones al eliminar la dependencia de precompiladores. Las aplicaciones se distribuyen no como código fuente de SQL incorporado, que se tiene que preprocesar para cada producto de base de datos, sino como aplicaciones compiladas o bibliotecas en tiempo de ejecución.
- Las aplicaciones CLI de DB2 individuales no se tienen que vincular a cada base de datos; solo los archivos de vinculación que se suministran con CLI de DB2 se tienen que vincular una vez para todas las aplicaciones CLI de DB2. Esto puede reducir significativamente la cantidad de gestión necesaria para la aplicación cuando ya está en nivel de uso general.
- Las aplicaciones CLI de DB2 se pueden conectar a varias bases de datos, incluidas varias conexiones a la misma base de datos, todo desde la misma aplicación. Cada conexión tiene su propio ámbito de confirmación. Esto resulta más sencillo si se utiliza CLI que si se utiliza SQL incorporado, donde la aplicación debe utilizar varias hebras para conseguir el mismo resultado.
- CLI de DB2 elimina la necesidad de áreas de datos controladas por la aplicación y a menudo complejas, como SQLDA y SQLCA, que suelen estar asociadas con aplicaciones de SQL incorporado. En su lugar, CLI de DB2 asigna y controla las

estructuras de datos necesarias y proporciona un *descriptor de contexto* para que la aplicación haga referencia a las mismas.

- CLI de DB2 permite el desarrollo de aplicaciones de varias hebras seguras en las que cada hebra puede tener su propia conexión y un ámbito de confirmación independiente del resto. CLI de DB2 lo consigue eliminando las áreas de datos descritas anteriormente y asociando todas estas estructuras de datos a las que puede acceder la aplicación con un descriptor de contexto específico. A diferencia de SQL incorporado, una aplicación CLI de varias hebras no tiene que llamar a ninguna de las API de DB2® de gestión de contexto; el controlador de CLI de DB2 lo maneja automáticamente.
- CLI de DB2 proporciona una función mejorada de captación y entrada de parámetros que permite especificar matrices de datos como entrada, recuperar varias filas de un conjunto de resultados directamente en una matriz y ejecutar sentencias que generan varios conjuntos de resultados.
- CLI de DB2 proporciona una interfaz coherente frente a la información de catálogo de consulta (Tablas, Columnas, Claves foráneas, Claves principales, etc.) contenida en distintas tablas de catálogo de DBMS. Los conjuntos de resultados devueltos son coherentes entre los DBMS. Esto protege la aplicación frente a cambios de catálogo entre releases de servidores de datos, así como frente a diferencias de catálogo entre distintos servidores de bases de datos; por lo tanto evita que las aplicaciones tengan que escribir consultas de catálogo específicas de cada versión y específicas de cada servidor.
- CLI de DB2 también proporciona conversión de datos ampliados, por lo que se necesita menos código de aplicación cuando se convierte información entre distintos tipos de datos SQL y C.
- CLI de DB2 incorpora funciones CLI tanto de ODBC como de X/Open (ambas son especificaciones aceptadas de la industria). CLI de DB2 también cumple con el estándar de CLI de ISO. El conocimiento que los programadores de aplicaciones adquieran en estas especificaciones se puede aplicar directamente al desarrollo de CLI de DB2 y viceversa. Esta interfaz resulta intuitiva para los programadores que están familiarizados con bibliotecas de funciones pero tienen pocos conocimientos sobre métodos específicos del producto para incorporar sentencias de SQL en un lenguaje principal.
- CLI de DB2 proporciona la posibilidad de recuperar varias filas y conjuntos de resultados generados a partir de un procedimiento almacenado que reside en un servidor DB2 Universal Database (o DB2 Universal Database para z/OS y OS/390 versión 5 o posterior). Sin embargo, tenga en cuenta que esta función existe para clientes de la Versión 5 de DB2 Universal Database que utilizan SQL incorporado si el procedimiento almacenado reside en un servidor al que se puede acceder desde un servidor DataJoiner® Versión 2.
- CLI de DB2 ofrece soporte más amplio para cursores desplazables. Con cursores desplazables puede desplazarse por un cursor del siguiente modo:
  - Hacia adelante, una o más filas
  - Hacia atrás, una o más filas
  - Desde la primera fila una o más filas.
  - Desde la última fila una o más filas.

  Los cursores desplazables se pueden utilizar junto con una salida de matriz. Puede declarar un cursor actualizable como desplazable y luego avanzar o retroceder una o más filas por el conjunto de resultados. También puede captar filas especificando un desplazamiento con respecto a:
  - La fila actual
  - El principio o fin del conjunto de resultados

– Una fila específica que ha establecido anteriormente con una marca.

Cuándo utilizar CLI de DB2 o SQL incorporado

La interfaz elegida depende de la aplicación.

CLI de DB2 resulta ideal para aplicaciones de interfaz gráfica de usuario (GUI) basada en consulta que necesitan portabilidad. Las ventajas explicadas anteriormente pueden hacer parecer que utilizar CLI de DB2 sea la opción obvia para cualquier aplicación. Sin embargo, hay un factor que se debe tener en cuenta: la comparación entre SQL estático y dinámico. Es mucho más fácil utilizar SQL estático en aplicaciones incorporadas.

SQL estático tiene varias ventajas:

- Rendimiento

  SQL dinámico se prepara en el tiempo de ejecución, SQL estático se prepara en el momento de la precompilación. Además de necesitar más proceso, el paso de preparación puede originar tráfico de red adicional en el tiempo de ejecución. El tráfico de red adicional se puede evitar si la aplicación CLI de DB2 utiliza la preparación diferida (que es el comportamiento por omisión).

  Es importante indicar que SQL estático no siempre tiene un rendimiento mejor que SQL dinámico. SQL dinámico se prepara en el tiempo de ejecución y utiliza las estadísticas de bases de datos disponibles en ese momento, mientras que SQL estático utiliza las estadísticas de bases de datos disponibles en el momento de realizar la operación BIND. SQL dinámico utiliza los cambios realizados en la base de datos, como nuevos índices, para elegir el plan de acceso óptimo, lo que potencialmente ofrece un mejor rendimiento que el mismo SQL ejecutado como SQL estático. Además, se puede evitar la precompilación de sentencias de SQL dinámico si se colocan en antememoria.
- Encapsulación y seguridad

  En SQL estático, las autorizaciones para acceder a objetos (como una tabla o una vista) están asociadas a un paquete y se validan en el momento de vincular el paquete. Esto significa que los administradores de bases de datos sólo tienen que otorgar autorización de ejecución sobre un determinado paquete a un conjunto de usuarios (encapsulando así sus privilegios en el paquete) sin tener que otorgarles acceso explícito a cada objeto de la base de datos. En SQL dinámico, las autorizaciones se validan en el tiempo de ejecución sentencia a sentencia; por lo tanto, se tiene que otorgar a los usuarios acceso explícito a cada objeto de la base de datos. Esto permite que estos usuarios accedan a partes del objeto a las que no tienen necesidad de acceder.
- SQL incorporado recibe soporte en otros lenguajes, además de C o C++.
- Para selecciones fijas de consulta, SQL incorporado resulta más sencillo.

Si una aplicación necesita las ventajas de ambas interfaces, se puede utilizar SQL estático dentro de una aplicación CLI de DB2 creando un procedimiento almacenado que contenga el SQL estático. Se llama al procedimiento almacenado desde dentro de una aplicación CLI de DB2 y se ejecuta en el servidor. Una vez creado el procedimiento almacenado, cualquier aplicación CLI de DB2 o ODBC puede llamarlo.

También se puede escribir una aplicación mixta que utilice tanto CLI de DB2 como SQL incorporado, aprovechando sus respectivas ventajas. En este caso, se utiliza CLI de DB2 para proporcionar la aplicación base, con módulos clave escritos utilizando SQL estático por razones de rendimiento o seguridad. Esto complica el

diseño de la aplicación y sólo se debe utilizar si los procedimientos almacenados no cumplen con los requisitos de la aplicación.

Por último, la decisión sobre cuándo utilizar cada interfaz se basará en preferencias individuales y en la experiencia más que en cualquier otro factor.

Conceptos relacionados:

- "Recurso de rastreo de CLI/ODBC/JDBC" en la página 494

Tareas relacionadas:

- "Preparing and executing SQL statements in CLI applications" en la publicación *CLI Guide and Reference, Volume 1*
- "Issuing SQL statements in CLI applications" en la publicación *CLI Guide and Reference, Volume 1*
- "Creating static SQL with CLI/ODBC/JDBC Static Profiling" en la publicación *CLI Guide and Reference, Volume 1*

Capítulo 6. Programación en C y C++

Consideraciones sobre programación en C/C++ 145
Secuencias tri-grafo para C y C++ 146
Archivos de entrada y de salida para C y C++ . . 146
Archivos include 147
Archivos include para C y C++ 147
Archivos include en C y C++ 149
Sentencias de SQL incorporado en C y C++ . . . 150
Variables del lenguaje principal en C y C++ . . . 152
Variables del lenguaje principal en C y C++ . . 152
Nombres de variables del lenguaje principal en C y C++ 153
Declaraciones de variables del lenguaje principal en C y C++ 154
Sintaxis de las variables numéricas del lenguaje principal en C y C++ 154
Sintaxis de las variables del lenguaje principal de tipo carácter fijas y terminadas en nulo en C y C++ 156
Sintaxis de las variables del lenguaje principal de tipo carácter de longitud variable en C o C++ 157
Variables de indicador en C y C++ 158
Variables gráficas del lenguaje principal en C y C++ 158
Sintaxis de la declaración gráfica de formatos gráficos de un solo gráfico y terminados en nulo en C y C++ 159
Sintaxis para la declaración gráfica del formato estructurado VARGRAPHIC en C o C++ . . . 160
Sintaxis de las variables del lenguaje principal de objeto grande (LOB) en C o C++ 161
Sintaxis de las variables del lenguaje principal de localizador de objeto grande (LOB) en C o C++ 163
Sintaxis de declaraciones de variables del lenguaje principal de referencia de archivos en C o C++ 164
Inicialización de variables del lenguaje principal en C y C++ 165
Expansión de macros en C 165
Soporte de estructuras del lenguaje principal en C y C++ 166
Tablas de indicadores en C y C++ 168
Series terminadas en nulo en C y C++ 169
Variables del lenguaje principal utilizadas como tipos de datos de puntero en C y C++ 171
Miembros de datos de clase utilizados como variables del lenguaje principal en C y C++ . . 171
Operadores de calificación y de miembro en C y C++ 173
Codificación de caracteres de varios bytes en C y C++ 173
Tipos de datos wchar_t y sqldbchar en C y C++ 174
Opción del precompilador WCHARTYPE en C y C++ 174
Consideraciones sobre EUC en japonés o chino tradicional y UCS-2 en C y C++ 177
Sección declare de SQL con variables del lenguaje principal para C y C++ 178
Consideraciones sobre tipos de datos para C y C++ 179
Tipos de datos SQL soportados en C y C++ . . 180
FOR BIT DATA en C y C++ 184
Tipos de datos C y C++ para procedimientos, funciones y métodos 184
Variables SQLSTATE y SQLCODE en C y C++ . . 186

Consideraciones sobre programación en C/C++

En los temas siguientes se tratan las consideraciones especiales sobre la programación en lenguajes principales. Se incluye información sobre las restricciones de lenguaje, los archivos include específicos del lenguaje principal, la incorporación de sentencias de SQL y los tipos de datos soportados para las variables del lenguaje principal.

Información relacionada:

- "Ejemplos de C" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Secuencias tri-grafo para C y C++

Algunos caracteres del juego de caracteres de C o C++ no están disponibles en todos los teclados. Dichos caracteres se pueden entrar en un programa fuente C o C++ utilizando una secuencia de tres caracteres denominada *tri-grafo*. Los tri-grafos no se reconocen en las sentencias de SQL. El precompilador reconoce los tri-grafos siguientes dentro de las declaraciones de variables del lenguaje principal:

Tri-grafo	Definición
??(	Corchete izquierdo '['
??)	Corchete derecho ']'
??<	Llave izquierda '{'
??>	Llave derecha '}'

Los tri-grafos restantes que se listan a continuación se pueden producir en cualquier lugar de un programa fuente C o C++:

Tri-grafo	Definición
??=	Almohadilla '#'
??/	Barra inclinada invertida '\'
??'	Signo de intercalación '^'
??!	Barra vertical '|'
??–	Tilde '~'

Archivos de entrada y de salida para C y C++

Por omisión, el archivo de entrada puede tener las extensiones siguientes:

.sqc Para los archivos C en todas las plataformas soportadas

.sqC Para archivos C++ en plataformas UNIX®

.sqx Para archivos C++ en sistemas operativos Windows®

Por omisión, los archivos de salida del precompilador correspondiente tienen las siguientes extensiones:

.c Para los archivos C en todas las plataformas soportadas

.C Para los archivos C++ en plataformas UNIX

.cxx Para archivos C++ en sistemas operativos Windows

Puede utilizar la opción de precompilación OUTPUT para alterar temporalmente el nombre y la vía de acceso del archivo fuente de salida modificado. Si utiliza las opciones de precompilación TARGET C o TARGET CPLUSPLUS, no es necesario que el archivo de entrada tenga ninguna extensión concreta.

Archivos include

Las siguientes secciones describen los archivos include correspondientes a C y C++.

Archivos include para C y C++

Los archivos include específicos del lenguaje principal (archivos de cabecera) para C y C++ tienen la extensión de archivo `.h`. A continuación se describen los archivos include destinados a su uso en las aplicaciones del usuario.

SQL (sql.h)
Este archivo incluye prototipos específicos del lenguaje para el vinculador, el precompilador y las API de recuperación de mensajes de error. También define constantes del sistema.

SQLADEF (sqladef.h)
Este archivo contiene prototipos de funciones utilizados por aplicaciones C y C++ precompiladas.

SQLAPREP (sqlaprep.h)
Este archivo contiene definiciones necesarias para escribir su propio precompilador.

SQLCA (sqlca.h)
Este archivo define la estructura del Área de comunicaciones de SQL (SQLCA). El SQLCA contiene variables que utiliza el gestor de bases de datos para proporcionar a una aplicación información de error sobre la ejecución de sentencias de SQL y llamadas a API.

SQLCLI (sqlcli.h)
Este archivo contiene los prototipos y constantes de funciones necesarios para escribir una aplicación de Interfaz de nivel de llamada (CLI de DB2). Las funciones de este archivo son comunes para la Interfaz de nivel de llamada de X/Open y el Nivel esencial de ODBC.

SQLCLI1 (sqlcli1.h)
Este archivo contiene los prototipos y las constantes de funciones necesarios para escribir una Interfaz de nivel de llamada (CLI de DB2) que hace uso de las características más avanzadas de la CLI de DB2. Muchas de las funciones de este archivo son comunes para la Interfaz de nivel de llamada de X/Open y el Nivel 1 de ODBC. Además, este archivo también incluye funciones sólo de X/Open y funciones específicas de DB2.

Este archivo incluye tanto `sqlcli.h` como `sqlext.h` (que contiene definiciones de API de Nivel 2 de ODBC).

SQLCODES (sqlcodes.h)
Este archivo define constantes para el campo SQLCODE de la estructura SQLCA.

SQLDA (sqlda.h)
Este archivo define la estructura del Área de descriptores de SQL (SQLDA). La SQLDA se utiliza para pasar datos entre una aplicación y el gestor de bases de datos.

SQLENV (sqlenv.h)
Este archivo define llamadas específicas del lenguaje para las API del entorno de bases de datos y las estructuras, constantes y códigos de retorno correspondientes a dichas interfaces.

SQLEXT (sqlext.h)
Este archivo contiene los prototipos y constantes de funciones de aquellas API de Nivel 1 y Nivel 2 de ODBC que no forman parte de la especificación de la Interfaz de nivel de llamada de X/Open y, por lo tanto, se utiliza con permiso de Microsoft Corporation.

SQLE819A (sqle819a.h)
Si la página de códigos de la base de datos es 819 (ISO Latin-1), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 500 (EBCDIC internacional) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLE819B (sqle819b.h)
Si la página de códigos de la base de datos es 819 (ISO Latin-1), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 037 (EBCDIC inglés de EE.UU.) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLE850A (sqle850a.h)
Si la página de códigos de la base de datos es 850 (ASCII Latin-1), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 500 (EBCDIC internacional) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLE850B (sqle850b.h)
Si la página de códigos de la base de datos es 850 (ASCII Latin-1), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 037 (EBCDIC inglés de EE.UU.) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLE932A (sqle932a.h)
Si la página de códigos de la base de datos es 932 (ASCII japonés), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 5035 (EBCDIC japonés) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLE932B (sqle932b.h)
Si la página de códigos de la base de datos es 932 (ASCII japonés), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 5026 (EBCDIC japonés) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLJACB (sqljacb.h)
Este archivo define constantes, estructuras y bloques de control correspondientes a la interfaz de DB2 Connect.

SQLMON (sqlmon.h)
Este archivo define llamadas específicas del lenguaje para las API del supervisor del sistema de bases de datos y las estructuras, constantes y códigos de retorno correspondientes a dichas interfaces.

SQLSTATE (sqlstate.h)
Este archivo define constantes correspondientes al campo SQLSTATE de la estructura SQLCA.

SQLSYSTM (sqlsystm.h)
Este archivo contiene definiciones específicas de la plataforma que utilizan las API y las estructuras de datos del gestor de bases de datos.

SQLUDF (sqludf.h)
Este archivo define constantes y estructuras de interfaces para escribir funciones definidas por el usuario (UDF).

SQLUTIL (sqlutil.h)
Este archivo define las llamadas específicas del lenguaje correspondientes a las API de programas de utilidad y las estructuras, constantes y códigos necesarios para dichas interfaces.

SQLUV (sqluv.h)
Este archivo define estructuras, constantes y prototipos para la API asíncrona de Registro de lecturas y para las API utilizadas por los proveedores de carga y descarga de tablas.

SQLUVEND (sqluvend.h)
Este archivo define estructuras, constantes y prototipos correspondientes a las API que utilizarán los proveedores de gestión de almacenamiento.

SQLXA (sqlxa.h)
Este archivo contiene prototipos y constantes de funciones utilizados por las aplicaciones que usan la Interfaz XA de X/Open.

Conceptos relacionados:

• "Archivos include en C y C++" en la página 149

Archivos include en C y C++

Existen dos métodos para incluir archivos: la sentencia EXEC SQL INCLUDE y la macro #include. El precompilador pasará por alto #include y sólo procesará los archivos incluidos mediante la sentencia EXEC SQL INCLUDE.

Para localizar los archivos incluidos mediante `EXEC SQL INCLUDE`, el precompilador C de DB2® busca en primer lugar en el directorio actual y, a continuación, en los directorios especificados por la variable de entorno DB2INCLUDE. Considere los ejemplos siguientes:

- `EXEC SQL INCLUDE payroll;`

  Si el archivo especificado en la sentencia INCLUDE no está encerrado entre comillas, tal como en el ejemplo anterior, el precompilador C busca `payroll.sqc`, y luego `payroll.h`, en cada uno de los directorios que mira. En los sistemas operativos UNIX®, el precompilador C++ busca `payroll.sqC`, luego `payroll.sqx`, luego `payroll.hpp` y luego `payroll.h` en cada uno de los directorios que mira. En los sistemas operativos Window® de 32 bits, el precompilador C++ busca sucesivamente `payroll.sqx`, `payroll.hpp` y `payroll.h` en cada uno de los directorios que examina.
- `EXEC SQL INCLUDE 'pay/payroll.h';`

  Si el nombre de archivo está encerrado entre comillas, como en el caso anterior, no se añade ninguna extensión al nombre.

  Si el nombre de archivo entrecomillado no contiene una vía de acceso absoluta, se utiliza el contenido de DB2INCLUDE para buscar el archivo, y se le añade como prefijo la vía de acceso especificada en el nombre de archivo de INCLUDE. Por ejemplo, en los sistemas basados en UNIX, si DB2INCLUDE se establece en '`/disk2:myfiles/c`', el precompilador C/C++ busca '`./pay/payroll.h`', luego '`/disk2/pay/payroll.h`' y finalmente '`./myfiles/c/pay/payroll.h`'. En los mensajes del precompilador se muestra la vía de acceso en que se encuentra realmente el archivo. En los sistemas operativos basados en Windows, sustituya las barras inclinadas invertidas (\) del ejemplo anterior por barras inclinadas.

Nota: El procesador de línea de mandatos coloca en antememoria el valor de DB2INCLUDE. Para cambiar el valor de DB2INCLUDE después de haber emitido mandatos del CLP, entre el mandato TERMINATE, luego vuelva a conectar con la base de datos y realice una precompilación del modo habitual.

Como ayuda para relacionar los errores del compilador con la fuente original, el precompilador genera macros ANSI #line en el archivo de salida. Esto permite que el compilador informe de los errores utilizando el nombre de archivo y el número de línea del archivo fuente o fuente incluido, en lugar de la salida del precompilador.

Sin embargo, si se especifica la opción PREPROCESSOR, todas las macros #line generadas por el precompilador hacen referencia al archivo preprocesado por el preprocesador C externo.

Algunos depuradores y otras herramientas que relacionan código fuente con código objeto no siempre funcionan bien con la macro #line. Si la herramienta que desea utilizar se comporta de forma inesperada, utilice la opción NOLINEMACRO (usada con DB2 PREP) cuando realice la precompilación. Esta opción evita que se generen macros #line.

Conceptos relacionados:

- "Expansión de macros en C" en la página 165

Información relacionada:

- "Sentencia PREPARE" en la publicación *Consulta de SQL, Volumen 2*
- "Archivos include para C y C++" en la página 147

Sentencias de SQL incorporado en C y C++

Las sentencias de SQL incorporado constan de los tres elementos siguientes:

Elemento	Sintaxis correcta
Inicializador de la sentencia	EXEC SQL
Serie de la sentencia	Cualquier sentencia de SQL válida
Terminador de la sentencia	punto y coma (;)

Por ejemplo:

```
EXEC SQL SELECT col INTO :hostvar FROM table;
```

Se aplican las normas siguientes a las sentencias de SQL incorporado:

- Puede empezar la serie de la sentencia de SQL en la misma línea que el par de palabras clave o en una línea separada. La serie de la sentencia puede tener una longitud de varias líneas. No divida el par de palabras clave EXEC SQL en distintas líneas.
- Debe utilizar el terminador de sentencias de SQL. De no utilizarlo, el precompilador seguirá hasta el siguiente terminador de la aplicación. Esto puede producir errores indeterminados.

  Se pueden colocar comentarios de C/C++ delante del inicializador de la sentencia o detrás del terminador de la sentencia.

- Se pueden poner varias sentencias de SQL y de C/C++ en la misma línea. Por ejemplo:

```
EXEC SQL OPEN c1; if (SQLCODE >= 0) EXEC SQL FETCH c1 INTO :hv;
```

- El precompilador SQL deja tal cual los retornos de carro, saltos de línea y tabulaciones que aparecen en una serie entrecomillada.
- Están permitidos los comentarios de SQL en cualquier línea que forme parte de una sentencia de SQL incorporado. Estos comentarios no están permitidos en las sentencias que se ejecutan de forma dinámica. El formato de un comentario de SQL consiste en un guión doble (`--`) seguido de una serie compuesta por cero o más caracteres y terminado por un fin de línea. No coloque comentarios de SQL después del terminador de sentencia de SQL. Los comentarios después del terminador generan errores de compilación porque parece que formen parte del lenguaje C/C++.

  Puede utilizar comentarios en la serie de una sentencia estática siempre que se permitan caracteres en blanco. Utilice los delimitadores de comentarios de C/C++ `/* */` o el símbolo de comentario de SQL (`--`). Los comentarios de C++ del estilo `//` no están permitidos en las sentencias de SQL estático, pero se pueden utilizar en cualquier otro lugar del programa. El precompilador elimina los comentarios antes de procesar la sentencia de SQL. **No puede** utilizar los delimitadores de comentario de C y C++ `/* */` ni `//` en una sentencia de SQL dinámico. Sin embargo, los puede utilizar en cualquier otro lugar del programa.
- Puede continuar los literales de la serie de SQL y los identificadores delimitados con divisiones de línea en las aplicaciones C y C++. Para ello, utilice una barra inclinada invertida (`\`) al final de la línea en que desea que se produzca la división. Por ejemplo:

```
EXEC SQL SELECT "NA\
ME" INTO :n FROM staff WHERE name='Sa\
nders';
```

  Los caracteres de línea nueva (como, por ejemplo, el retorno de carro y el salto de línea) no se incluyen en la serie ni en el identificador delimitado.
- La sustitución de caracteres de espacio en blanco, como caracteres de fin de línea y de tabulación, se produce del siguiente modo:
  - Cuando aparecen fuera de las comillas (pero dentro de sentencias de SQL), los caracteres de fin de línea y tabuladores se sustituyen por un solo espacio.
  - Si aparecen dentro de las comillas, los caracteres de fin de línea desaparecen, siempre que se continúe correctamente la serie para un programa C. Los tabuladores no se modifican.

  Observe que los caracteres reales utilizados para el final de línea y el tabulador varían según la plataforma. Por ejemplo, los sistemas basados en UNIX® utilizan un carácter de salto de línea.

Información relacionada:

- Apéndice A, “Sentencias de SQL soportadas”, en la página 733

Variables del lenguaje principal en C y C++

Las secciones siguientes describen cómo declarar y utilizar variables del lenguaje principal en programas C y C++.

Variables del lenguaje principal en C y C++

Las variables del lenguaje principal son variables de los lenguajes C o C++ a las que se hace referencia en las sentencias de SQL. Permiten que una aplicación pase datos de entrada al gestor de bases de datos y reciba datos de salida de éste. Una vez que se ha precompilado la aplicación, el compilador utiliza las variables del lenguaje principal como cualquier otra variable de C/C++. Cuando denomine, declare y utilice variables del lenguaje principal, siga las normas descritas en los apartados siguientes.

En aplicaciones que construyen de la SQLDA de forma manual, no se pueden utilizar variables tipo long cuando sqlvar::sqltype==SQL_TYP_INTEGER. En su lugar se deben utilizar tipos sqlint32. Este problema es idéntico a utilizar variables long en declaraciones de variables del lenguaje principal, excepto en que con una SQLDA construida de forma manual el precompilador no revelará este error y se producirán errores en el tiempo de ejecución.

Las conversiones de long y unsigned long que se utilizan para acceder a información sqlvar::sqldata se deben cambiar por sqlint32 y sqluint32. Los miembros val correspondientes a las estructuras sqloptions y sqla_option se declaran como sqluintptr. Por lo tanto, la asignación de miembros de puntero en miembros sqla_option::val o sqloptions::val deben utilizar conversiones sqluintptr en lugar de conversiones unsigned long. Este cambio no ocasionará problemas en el tiempo de ejecución en plataformas UNIX® de 64 bits, pero se debe realizar como preparación para aplicaciones Windows® NT de 64 bits, en las que el tipo long sólo es de 32 bits.

Conceptos relacionados:

- “Nombres de variables del lenguaje principal en C y C++” en la página 153
- “Declaraciones de variables del lenguaje principal en C y C++” en la página 154
- “Sintaxis de las variables del lenguaje principal de tipo carácter fijas y terminadas en nulo en C y C++” en la página 156
- “Variables de indicador en C y C++” en la página 158
- “Variables gráficas del lenguaje principal en C y C++” en la página 158
- “Inicialización de variables del lenguaje principal en C y C++” en la página 165
- “Soporte de estructuras del lenguaje principal en C y C++” en la página 166
- “Sección declare de SQL con variables del lenguaje principal para C y C++” en la página 178

Información relacionada:

- “Sintaxis de las variables numéricas del lenguaje principal en C y C++” en la página 154
- “Sintaxis de las variables del lenguaje principal de tipo carácter de longitud variable en C o C++” en la página 157
- “Sintaxis de las variables del lenguaje principal de objeto grande (LOB) en C o C++” en la página 161
- “Sintaxis de las variables del lenguaje principal de localizador de objeto grande (LOB) en C o C++” en la página 163

- "Sintaxis de declaraciones de variables del lenguaje principal de referencia de archivos en C o C++" en la página 164

Nombres de variables del lenguaje principal en C y C++

El precompilador SQL identifica las variables del lenguaje principal por el nombre declarado para éstas. Se aplican las normas siguientes:

- Especifique nombres de variables con una longitud máxima de 255 caracteres.
- Empiece los nombres de variables con prefijos que no sean `SQL`, `sql`, `DB2` ni `db2`, los cuales están reservados para uso del sistema. Por ejemplo:

```
EXEC SQL BEGIN DECLARE SECTION;
  char  varsql;     /* permitido */
  char  sqlvar;     /* no permitido */
  char  SQL_VAR;    /* no permitido */
EXEC SQL END DECLARE SECTION;
```

- El precompilador considera los nombres de variables del lenguaje principal como globales respecto a un módulo. Sin embargo, esto no significa que las variables del lenguaje principal se tengan que definir como variables globales; es perfectamente aceptable que se declaren variables del lenguaje principal como variables locales dentro de funciones. Por ejemplo, el código siguiente funcionará correctamente:

```
void f1(int i)
{
EXEC SQL BEGIN DECLARE SECTION;
  short host_var_1;
EXEC SQL END DECLARE SECTION;
EXEC SQL SELECT COL1 INTO :host_var_1 from TBL1;
}
void f2(int i)
{
EXEC SQL BEGIN DECLARE SECTION;
  short host_var_2;
EXEC SQL END DECLARE SECTION;
EXEC SQL INSERT INTO TBL1 VALUES (:host_var_2);
}
```

  También es posible tener varias variables del lenguaje principal locales con el mismo nombre, siempre que sean del mismo tipo y tamaño. Para ello, declare la primera aparición de la variable del lenguaje principal, ante el precompilador, entre sentencias BEGIN DECLARE SECTION y END DECLARE SECTION, y deje las declaraciones posteriores de la variable fuente de las secciones de declaración. El código siguiente muestra un ejemplo de este caso:

```
void f3(int i)
{
EXEC SQL BEGIN DECLARE SECTION;
  char host_var_3[25];
EXEC SQL END DECLARE SECTION;
EXEC SQL SELECT COL2 INTO :host_var_3 FROM TBL2;
}
void f4(int i)
{
char host_var_3[25];
EXEC SQL INSERT INTO TBL2 VALUES (:host_var_3);
}
```

  Puesto que `f3` y `f4` están en el mismo módulo y `host_var_3` tiene el mismo tipo y longitud en ambas funciones, basta una sola declaración ante el precompilador para utilizarla en ambos lugares.

Conceptos relacionados:

• “Declaraciones de variables del lenguaje principal en C y C++” en la página 154

Declaraciones de variables del lenguaje principal en C y C++

Se debe utilizar una sección de declaración de SQL para identificar las declaraciones de variables del lenguaje principal. Esto alerta al precompilador acerca de las variables del lenguaje principal a las que se puede hacer referencia en sentencias de SQL posteriores.

El precompilador C/C++ sólo reconoce un subconjunto de declaraciones de C o C++ válidas como declaraciones de variables del lenguaje principal válidas. Estas declaraciones definen variables numéricas o de tipo carácter. No se admiten las definiciones de tipo para los tipos de variable del lenguaje principal. Las variables del lenguaje principal se pueden agrupar en una sola estructura de lenguaje principal. Puede declarar miembros de datos de clase C++ como variables del lenguaje principal.

Una variable numérica del lenguaje principal se puede utilizar como variable de entrada o de salida para cualquier valor numérico de entrada o salida de SQL. Una variable del lenguaje principal de tipo carácter se puede utilizar como variable de entrada o de salida para cualquier valor de tipo carácter, de fecha, de hora o de indicación horaria, de entrada o salida de SQL. La aplicación se tiene que asegurar de que las variables de salida sean lo suficientemente largas como para contener los valores que reciben.

Conceptos relacionados:

- “Sintaxis de las variables del lenguaje principal de tipo carácter fijas y terminadas en nulo en C y C++” en la página 156
- “Variables gráficas del lenguaje principal en C y C++” en la página 158
- “Soporte de estructuras del lenguaje principal en C y C++” en la página 166
- “Miembros de datos de clase utilizados como variables del lenguaje principal en C y C++” en la página 171

Tareas relacionadas:

- “Declaración de variables del lenguaje principal con el Generador de declaraciones db2dclgn” en la página 32
- “Declaración de variables del lenguaje principal de tipo estructurado” en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*

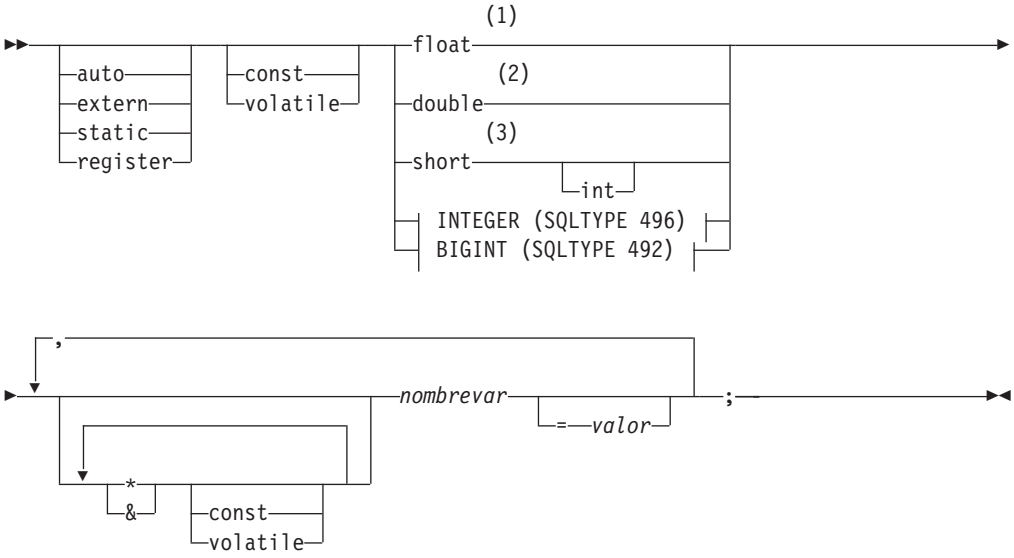
Información relacionada:

- “Sintaxis de las variables numéricas del lenguaje principal en C y C++” en la página 154
- “Sintaxis de las variables del lenguaje principal de tipo carácter de longitud variable en C o C++” en la página 157

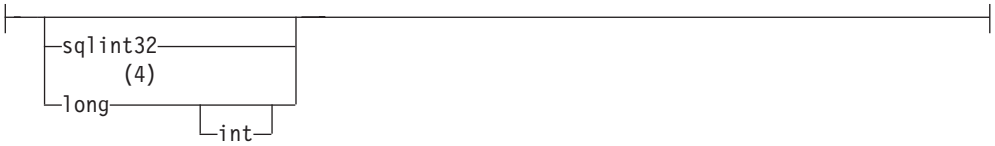
Sintaxis de las variables numéricas del lenguaje principal en C y C++

A continuación se muestra la sintaxis para declarar variables numéricas del lenguaje principal en C o C++.

Sintaxis de las variables numéricas del lenguaje principal en C o C++



INTEGER (SQLTYPE 496)



BIGINT (SQLTYPE 492)

Notas:

1 REAL (SQLTYPE 480), longitud 4

2 DOUBLE (SQLTYPE 480), longitud 8

3 SMALLINT (SQLTYPE 500)

4 Para obtener una portabilidad máxima de las aplicaciones utilice sqlint32 y sqlint64 respectivamente para las variables del lenguaje principal INTEGER y BIGINT. Por omisión, el uso de variables del lenguaje principal tipo long tiene como consecuencia el error del precompilador SQL0402 en las plataformas en que la longitud es una cantidad de 64 bits, como por ejemplo en UNIX de 64 BITS. Utilice la opción LONGERROR NO de PREP para imponer que DB2 acepte variables long como tipos de variable del lenguaje principal aceptables y las trate como variables BIGINT.

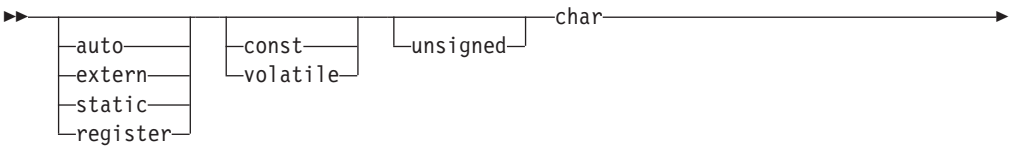
5 Para obtener una portabilidad máxima de las aplicaciones utilice,

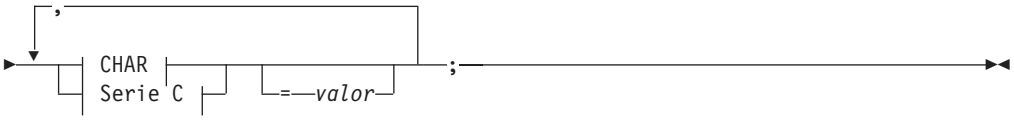
respectivamente, sqlint32 y sqlint64 para las variables del lenguaje principal INTEGER y BIGINT. Para utilizar el tipo de datos BIGINT, la plataforma tiene que soportar valores enteros de 64 bits. Por omisión, el uso de variables del lenguaje principal tipo long tiene como consecuencia el error del precompilador SQL0402 en las plataformas en que la longitud es una cantidad de 64 bits, como por ejemplo en UNIX de 64 BITS. Utilice la opción LONGERROR NO de PREP para imponer que DB2 acepte variables long como tipos de variable del lenguaje principal aceptables y las trate como variables BIGINT.

Sintaxis de las variables del lenguaje principal de tipo carácter fijas y terminadas en nulo en C y C++

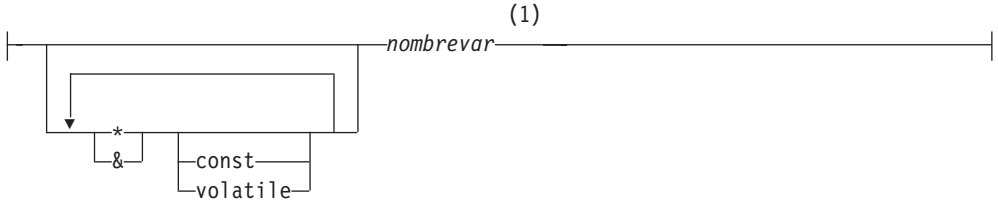
A continuación se muestra la sintaxis para declarar variables del lenguaje principal de tipo carácter fijas y terminadas en nulo en C o C++.

Sintaxis de las variables del lenguaje principal de tipo carácter fijas y terminadas en nulo en C

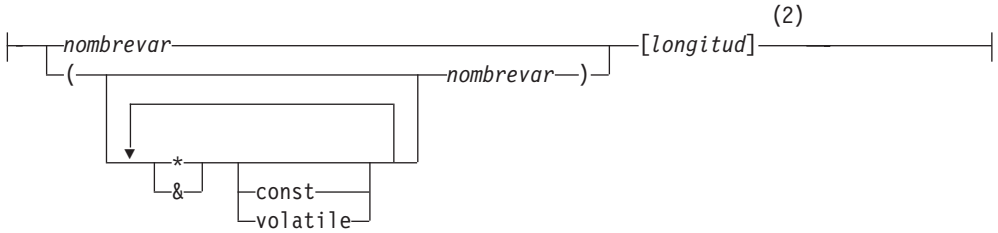




CHAR



Serie C



Notas:

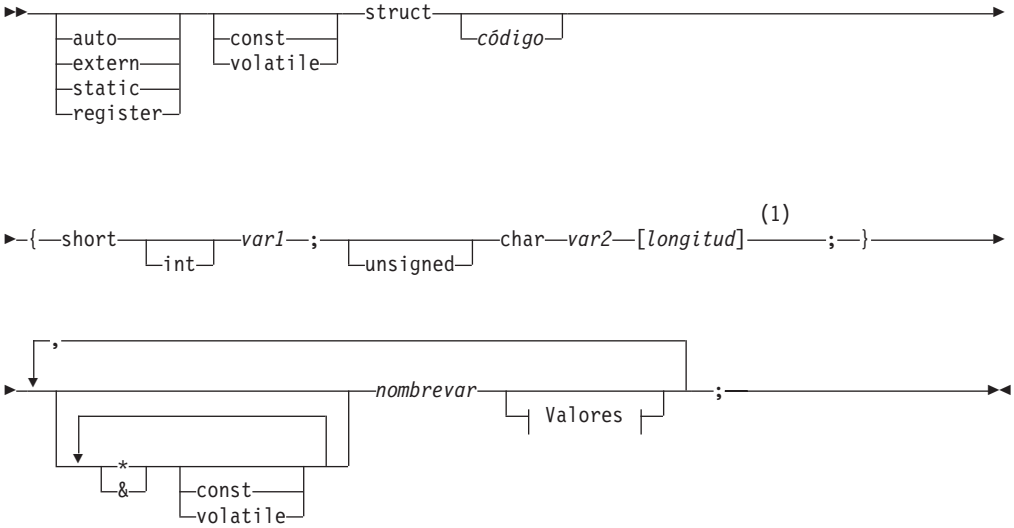
1 CHAR (SQLTYPE 452), length 1

2 Serie C terminada en nulo (SQLTYPE 460); la longitud puede ser cualquier expresión constante válida

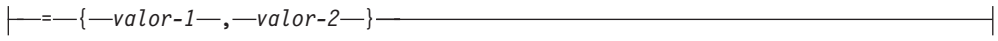
Sintaxis de las variables del lenguaje principal de tipo carácter de longitud variable en C o C++

A continuación se muestra la sintaxis para declarar variables del lenguaje principal de tipo carácter de longitud variable en C o C++.

Sintaxis para variables del lenguaje principal de tipo carácter de longitud variable en C



Valores



Notas:

1 En el formato 2, la longitud puede ser cualquier expresión constante válida. Su valor después de una evaluación determina si la variable del lenguaje principal es VARCHAR (SQLTYPE 448) o LONG VARCHAR (SQLTYPE 456).

Consideraciones sobre las variables del lenguaje principal de tipo carácter de longitud variable:

1. Aunque el gestor de bases de datos convierte los datos de tipo carácter al **formato 1** o al **formato 2** siempre que es posible, el **formato 1** corresponde a los tipos de columna CHAR o VARCHAR, mientras que el **formato 2** corresponde a los tipos de columna VARCHAR y LONG VARCHAR.
2. Si se utiliza el **formato 1** con un especificador de longitud *[n]*, el valor correspondiente al especificador de longitud tras la evaluación no debe ser mayor que 32672 y la serie contenida en la variable debe terminar en nulo.
3. Si se utiliza el **formato 2**, el valor correspondiente al especificador de longitud tras la evaluación no debe ser mayor que 32700.
4. En el **formato 2**, *var1* y *var2* deben ser simples referencias a variables (y no operadores) y no se pueden utilizar como variables del lenguaje principal (*nombrevar* es la variable del lenguaje principal).

5. *nombrevar* puede ser un simple nombre de variable o puede incluir operadores, como por ejemplo **nombrevar*. Consulte la descripción de los tipos de datos de puntero en C y C++ para obtener más información.
6. El precompilador determina el SQLTYPE y la SQLLEN de todas las variables del lenguaje principal. Si una variable del lenguaje principal aparece en una sentencia de SQL con una variable de indicador, mientras dure esa sentencia se asigna como SQLTYPE el SQLTYPE base más uno.
7. El precompilador permite algunas declaraciones que no son válidas sintácticamente en C ni en C++. Si tiene dudas acerca de la sintaxis de una declaración determinada, consulte la documentación del compilador.

Conceptos relacionados:

- "Variables del lenguaje principal utilizadas como tipos de datos de puntero en C y C++" en la página 171

Variables de indicador en C y C++

Las variables de indicador se deben declarar con tipo de datos `short`.

Conceptos relacionados:

- "Tablas de indicadores en C y C++" en la página 168

Variables gráficas del lenguaje principal en C y C++

Para manejar datos gráficos en aplicaciones C o C++, utilice variables de lenguaje principal basadas en el tipo de datos `wchar_t` de C/C++ o en el tipo de datos `sqldbchar` proporcionado por DB2®. Puede asignar estos tipos de variables del lenguaje principal a las columnas de una tabla que sean GRAPHIC, VARGRAPHIC o DBCLOB. Por ejemplo, puede actualizar o seleccionar datos DBCS de las columnas GRAPHIC o VARGRAPHIC de una tabla.

Existen tres formatos válidos para una variable gráfica del lenguaje principal:

- Formato de gráfico simple

  Las variables de gráfico simple del lenguaje principal tienen para SQLTYPE el valor 468/469, lo que equivale al tipo de datos GRAPHIC(1) de SQL.
- Formato de gráfico terminado en nulo

  Terminado en nulo hace referencia a una situación en que todos los bytes del último carácter de la serie gráfica contienen ceros binarios ('\0's). Su tipo de datos de SQL es 400®/401.
- Formato estructurado VARGRAPHIC

  Las variables de formato estructurado VARGRAPHIC del lenguaje principal tienen para SQLTYPE el valor 464/465 si su longitud está comprendida entre 1 y 16.336 bytes. Tienen para SQLTYPE el valor 472/473 si su longitud está comprendida entre 2.000 y 16.350 bytes.

Conceptos relacionados:

- "Nombres de variables del lenguaje principal en C y C++" en la página 153
- "Declaraciones de variables del lenguaje principal en C y C++" en la página 154
- "Inicialización de variables del lenguaje principal en C y C++" en la página 165
- "Soporte de estructuras del lenguaje principal en C y C++" en la página 166
- "Tablas de indicadores en C y C++" en la página 168

- "Codificación de caracteres de varios bytes en C y C++" en la página 173
- "Tipos de datos wchar_t y sqldbchar en C y C++" en la página 174
- "Opción del precompilador WCHARTYPE en C y C++" en la página 174

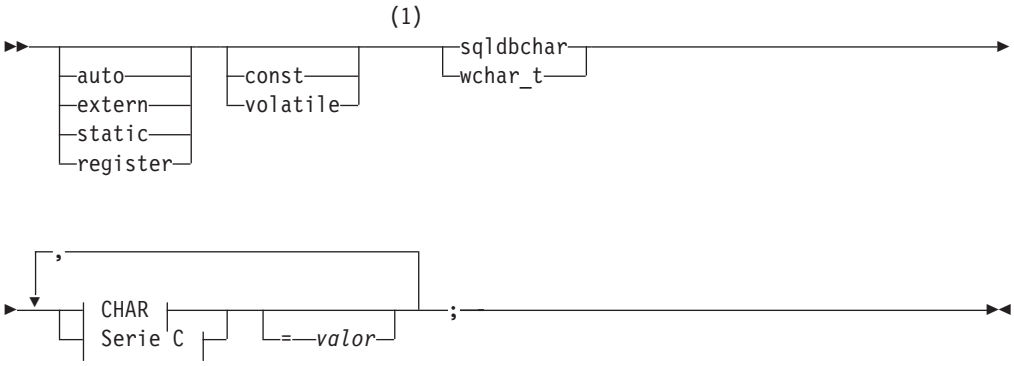
Información relacionada:

- "Sintaxis de la declaración gráfica de formatos gráficos de un solo gráfico y terminados en nulo en C y C++" en la página 159
- "Sintaxis para la declaración gráfica del formato estructurado VARGRAPHIC en C o C++" en la página 160
- "Sintaxis de las variables del lenguaje principal de objeto grande (LOB) en C o C++" en la página 161
- "Sintaxis de las variables del lenguaje principal de localizador de objeto grande (LOB) en C o C++" en la página 163
- "Sintaxis de declaraciones de variables del lenguaje principal de referencia de archivos en C o C++" en la página 164

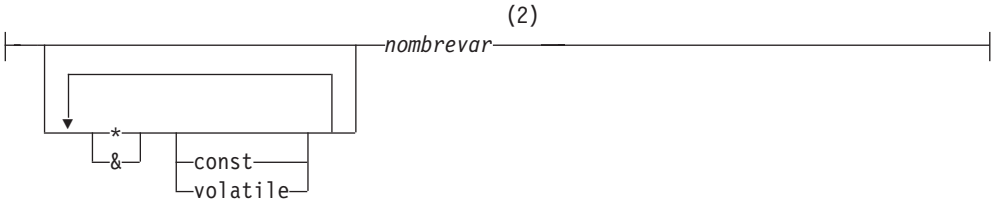
Sintaxis de la declaración gráfica de formatos gráficos de un solo gráfico y terminados en nulo en C y C++

A continuación se muestra la sintaxis para declarar una variable del lenguaje principal gráfica mediante el formato de un solo gráfico y el formato de gráfico terminado en nulo.

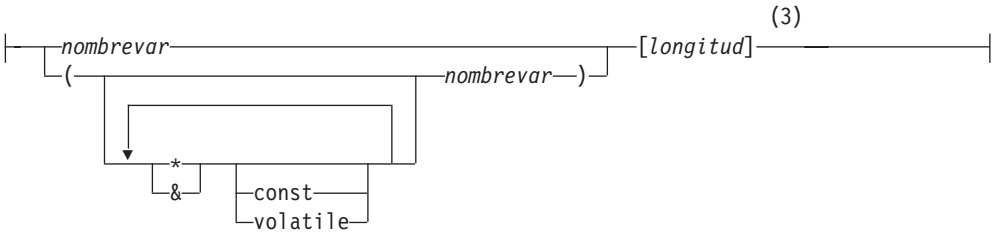
Sintaxis para la declaración gráfica de formato de un solo gráfico y



CHAR



Serie C



Notas:

1 Para determinar cuál de los dos tipos gráficos debe utilizar, consulte la descripción de los tipos de datos wchar_t y sqldbchar en C y C++.

2 GRAPHIC (SQLTYPE 468), longitud 1

3 Serie gráfica terminada en nulo (SQLTYPE 400)

Consideraciones sobre las variables del lenguaje principal de gráficos:

1. El formato de gráfico simple declara una variable del lenguaje principal de serie gráfica y longitud fija de longitud 1 con un SQLTYPE de 468 ó 469.
2. *valor* es un inicializador. Se debe utilizar un literal de serie de caracteres amplios (literal L) si se utiliza la opción WCHARTYPE CONVERT del precompilador.
3. *longitud* puede ser cualquier expresión constante válida, y su valor después de una evaluación tiene que ser mayor o igual a 1 y no mayor que la longitud máxima de VARGRAPHIC, que es 16336.
4. Las series gráficas terminadas en nulo se manejan de forma distinta en función del valor del establecimiento de opciones de precompilación de nivel estándar.

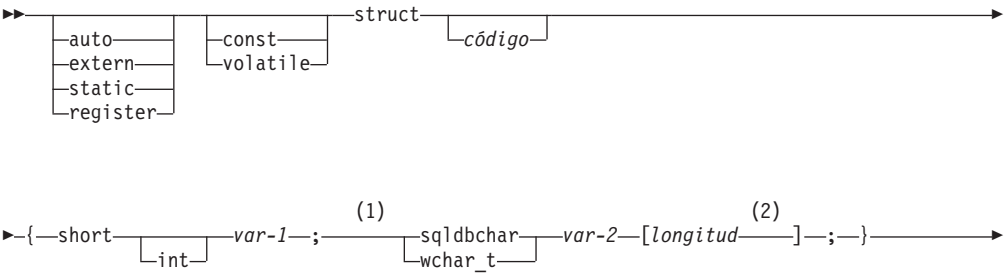
Conceptos relacionados:

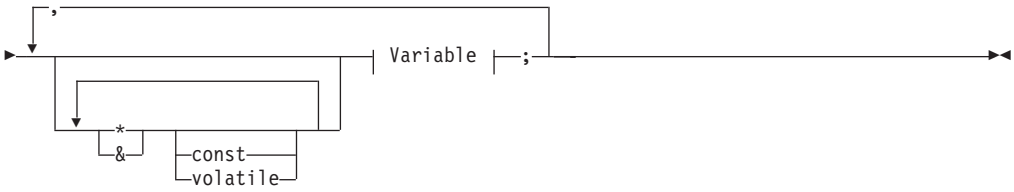
- "Series terminadas en nulo en C y C++" en la página 169
- "Tipos de datos wchar_t y sqldbchar en C y C++" en la página 174

Sintaxis para la declaración gráfica del formato estructurado VARGRAPHIC en C o C++

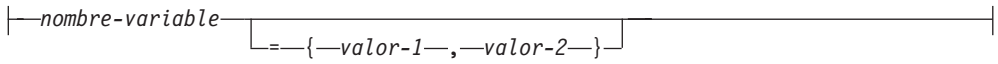
A continuación se muestra la sintaxis para declarar una variable del lenguaje principal gráfica mediante el formato estructurado VARGRAPHIC.

Sintaxis para la declaración gráfica del formato estructurado VARGRAPHIC





Variable:



Notas:

1 Para determinar cuál de los dos tipos gráficos debe utilizar, consulte la descripción de los tipos de datos wchar_t y sqldbchar en C y C++.

2 *longitud* puede ser cualquier expresión de constante válida. Su valor después de una evaluación determina si la variable del lenguaje principal es VARGRAPHIC (SQLTYPE 464) o LONG VARGRAPHIC (SQLTYPE 472). El valor de longitud debe ser mayor o igual que 1 y menor o igual que la longitud máxima de LONG VARGRAPHIC, que es 16350.

Consideraciones sobre la declaración gráfica (formato estructurado

VARGRAPHIC):

1. *var-1* y *var-2* deben ser simples referencias a variables (y no operadores) y no se pueden utilizar como variables del lenguaje principal.
2. *valor-1* y *valor-2* son inicializadores de *var-1* y *var-2*.*valor-1* debe ser un entero y *valor-2* debe ser un literal de serie de caracteres amplios (literal L) si se utiliza la opción WCHARTYPE CONVERT del precompilador.
3. Se puede utilizar el *código* struct para definir otras áreas de datos, pero no se puede utilizar por sí mismo como variable del lenguaje principal.

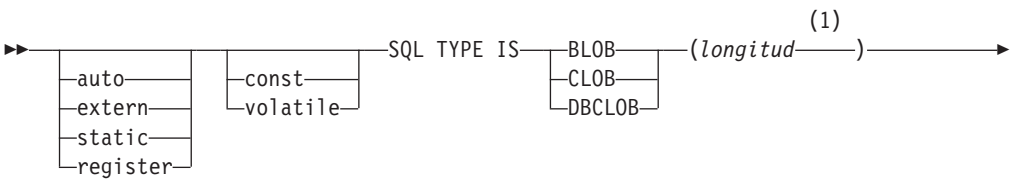
Conceptos relacionados:

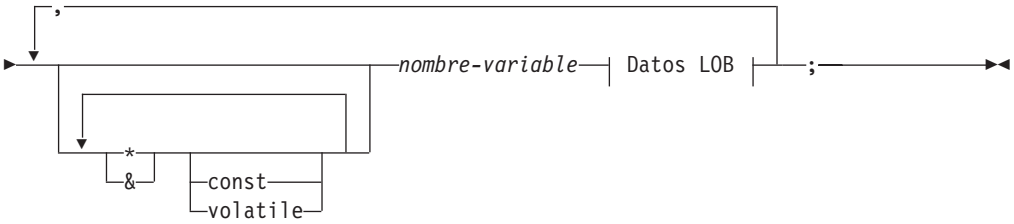
- "Tipos de datos wchar_t y sqldbchar en C y C++" en la página 174

Sintaxis de las variables del lenguaje principal de objeto grande (LOB) en C o C++

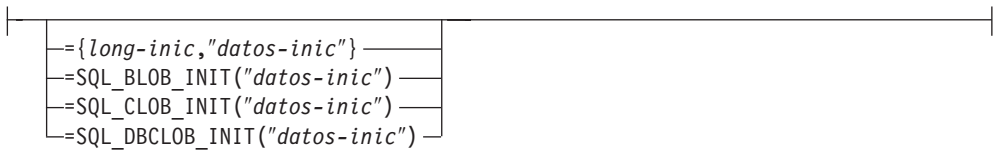
A continuación se muestra la sintaxis para declarar variables del lenguaje principal de objeto grande (LOB) en C o C++.

Sintaxis de las variables del lenguaje principal





Datos LOB



Notas:

1 *longitud* puede ser cualquier expresión de constante válida en la que se pueden utilizar las constantes K, M o G. El valor de la longitud después de una evaluación para BLOB y CLOB debe ser 1 <= longitud <= 2.147.483.647. El valor de *longitud* tras la evaluación correspondiente a DBCLOB debe ser 1 <= longitud <= 1.073.741.823.

Consideraciones sobre las variables del lenguaje principal LOB:

1. Es necesaria la cláusula TYPE IS de SQL para poder distinguir los tres tipos de LOB entre sí, de forma que se puedan llevar a cabo una comprobación del tipo y una resolución de la función para las variables de lenguaje principal de tipo LOB que se pasan a las funciones.
2. SQL TYPE IS, BLOB, CLOB, DBCLOB, K, M, G se pueden especificar en una mezcla de mayúsculas y minúsculas.
3. La longitud máxima permitida para la serie de inicialización *"datos-inic"* es 32702 bytes, incluidos los delimitadores de la serie (igual al límite existente en series C/C++ dentro del precompilador).
4. La longitud de inicialización, *long-inic*, tiene que ser una constante numérica (esto es, no puede incluir K, M ni G).
5. Se debe especificar una longitud para el LOB; es decir, que la declaración siguiente no está permitida:

   ```
   SQL TYPE IS BLOB my_blob;
   ```

6. Si el LOB no se inicializa dentro de la declaración, no se realizará ninguna inicialización en el código generado por el precompilador.
7. Si se inicializa un DBCLOB, es responsabilidad del usuario añadirle a la serie el prefijo 'L' (que indica una serie de caracteres amplios).

   Nota: Los literales de caracteres amplios, por ejemplo `L"Hello"`, sólo se deben utilizar en un programa precompilado si se selecciona la opción WCHARTYPE CONVERT de precompilación.
8. El precompilador genera un código de estructura que se puede utilizar para pasarlo al tipo de la variable del lenguaje principal.

Ejemplo de BLOB:

La declaración:

```
static Sql Type is Blob(2M) my_blob=SQL_BLOB_INIT("mydata");
```

Tiene como resultado la generación de la estructura siguiente:

```
static struct my_blob_t {
      sqluint32        length;
      char             data[2097152];
 } my_blob=SQL_BLOB_INIT("mydata");
```

Ejemplo de CLOB:

La declaración:

```
volatile sql type is clob(125m) *var1, var2 = {10, "data5data5"};
```

Tiene como resultado la generación de la estructura siguiente:

```
volatile struct var1_t {
     sqluint32        length;
     char             data[131072000];
} * var1, var2 = {10, "data5data5"};
```

Ejemplo de DBCLOB:

La declaración:

```
SQL TYPE IS DBCLOB(30000) my_dbclob1;
```

Si se precompila con la opción WCHARTYPE NOCONVERT, tiene como resultado la generación de la estructura siguiente:

```
struct my_dbclob1_t {
     sqluint32        length;
     sqldbchar        data[30000];
} my_dbclob1;
```

La declaración:

```
SQL TYPE IS DBCLOB(30000) my_dbclob2 = SQL_DBCLOB_INIT(L"mydbdata");
```

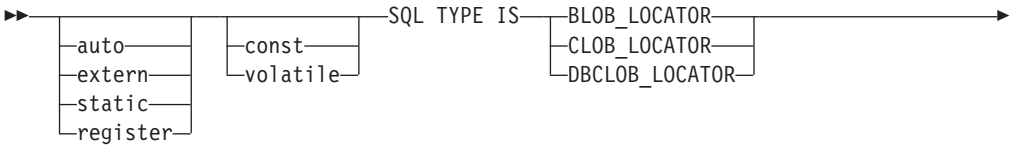
Si se precompila con la opción WCHARTYPE CONVERT, tiene como resultado la generación de la estructura siguiente:

```
struct my_dbclob2_t {
     sqluint32        length;
     wchar_t          data[30000];
} my_dbclob2 = SQL_DBCLOB_INIT(L"mydbdata");
```

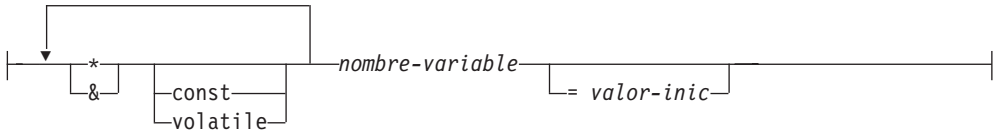
Sintaxis de las variables del lenguaje principal de localizador de objeto grande (LOB) en C o C++

A continuación se muestra la sintaxis para declarar variables del lenguaje principal de localizador de objeto grande (LOB) en C o C++.

Sintaxis de las variables del lenguaje principal de localizador de objeto grande (LOB)



Variable



Consideraciones sobre las variables del lenguaje principal de localizador de LOB:

1. SQL TYPE IS, BLOB_LOCATOR, CLOB_LOCATOR, DBCLOB_LOCATOR se pueden especificar en una mezcla de mayúsculas y minúsculas.
2. *valor-inic* permite la inicialización de variables de localizador de puntero y referencia. Otros tipos de inicialización no tendrán ningún significado.

Ejemplo de localizador de CLOB (existen otras declaraciones de tipos de localizador de LOB parecidas):

La declaración:

```
SQL TYPE IS CLOB_LOCATOR my_locator;
```

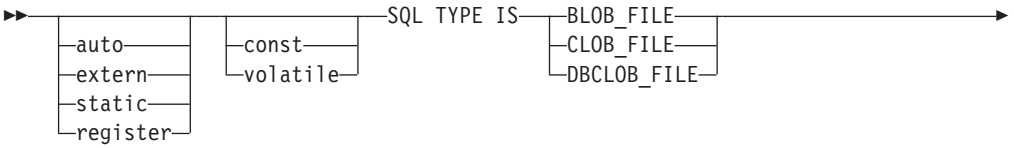
Tiene como resultado la generación de la declaración siguiente:

```
sqluint32 my_locator;
```

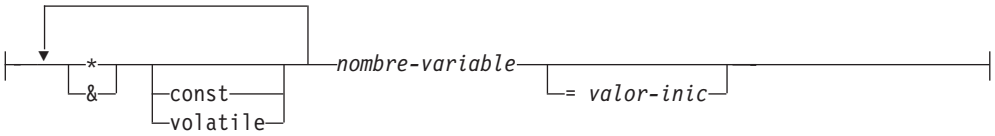
Sintaxis de declaraciones de variables del lenguaje principal de referencia de archivos en C o C++

A continuación se muestra la sintaxis para declarar variables del lenguaje principal de referencia de archivos en C o C++.

Sintaxis de las variables del lenguaje principal de referencia de archivos en C o C++



Variable



Nota: SQL TYPE IS, BLOB_FILE, CLOB_FILE, DBCLOB_FILE se pueden especificar en una mezcla de mayúsculas y minúsculas.

Ejemplo de referencia de archivos CLOB (existen otras declaraciones de tipo de referencia de archivos LOB parecidas):

La declaración:

```
static volatile SQL TYPE IS BLOB_FILE my_file;
```

Tiene como resultado la generación de la estructura siguiente:

```
static volatile struct {
     sqluint32        name_length;
     sqluint32        data_length;
     sqluint32        file_options;
              char    name[255];
} my_file;
```

Inicialización de variables del lenguaje principal en C y C++

En las secciones declare de C++ no se pueden inicializar variables del lenguaje principal mediante paréntesis. El ejemplo siguiente muestra los métodos correcto e incorrecto de inicialización en una sección de declaración:

```
EXEC SQL BEGIN DECLARE SECTION;
  short my_short_2 = 5;       /* correcto   */
  short my_short_1(5);        /* incorrecto */
EXEC SQL END DECLARE SECTION;
```

Expansión de macros en C

El precompilador C/C++ no puede procesar directamente ninguna macro de C utilizada en una declaración dentro de una sección de declaración. En lugar de esto, en primer lugar se debe preprocesar el archivo fuente mediante un preprocesador de C externo. Para ello, especifique el mandato exacto para invocar un preprocesador de C para el precompilador mediante la opción PREPROCESSOR.

Cuando se especifica la opción PREPROCESSOR, el precompilador procesa en primer lugar todas las sentencias de SQL INCLUDE, incorporando al archivo fuente el contenido de todos los archivos a los que se hace referencia en la sentencia de SQL INCLUDE. A continuación, el precompilador invoca al preprocesador de C externo, utilizando el mandato que se especifique con el archivo fuente modificado como entrada. El archivo preprocesado, del cual el precompilador siempre espera que tenga la extensión `.i`, se utiliza como el nuevo archivo fuente para el resto del proceso de precompilación.

Cualquier macro #line generada por el precompilador deja de hacer referencia al archivo fuente original, pasando a hacerla al archivo preprocesado. Para relacionar

los errores del compilador con el archivo fuente original, mantenga comentarios en el archivo preprocesado. Le serán de ayuda para localizar diversas secciones de los archivos fuente originales, incluidos los archivos de cabecera. La opción para mantener comentarios suele estar disponible en los preprocesadores C y la puede incluir en el mandato que especifique mediante la opción PREPROCESSOR. No debe hacer que el preprocesador de C produzca por sí mismo como salida ninguna macro #line, puesto que se podrían mezclar incorrectamente con las generadas por el precompilador.

Notas sobre la utilización de la expansión de macros:

1. El mandato que especifique mediante la opción PREPROCESSOR debe incluir todas las opciones deseadas, pero no el nombre del archivo de entrada. Por ejemplo, para C de IBM® en AIX® puede utilizar la opción:
   ```
   xlC -P -DMYMACRO=1
   ```
2. El precompilador espera que el mandato genere un archivo preprocesado con la extensión `.i`. Sin embargo, no se puede utilizar la redirección para generar el archivo preprocesado. Por ejemplo, **no se puede** utilizar la opción siguiente para generar un archivo preprocesado:
   ```
   xlC -E > x.i
   ```
3. Los errores que encuentre el preprocesador de C externo se notifican en un archivo que tiene el nombre correspondiente al archivo fuente original pero con la extensión `.err`.

Por ejemplo, puede utilizar la expansión de macros en el código fuente tal como se indica a continuación:

```
#define SIZE 3

EXEC SQL BEGIN DECLARE SECTION;
  char a[SIZE+1];
  char b[(SIZE+1)*3];
  struct
  {
    short length;
    char  data[SIZE*6];
  } m;
  SQL TYPE IS BLOB(SIZE+1) x;
  SQL TYPE IS CLOB((SIZE+2)*3) y;
  SQL TYPE IS DBCLOB(SIZE*2K) z;
EXEC SQL END DECLARE SECTION;
```

Las declaraciones anteriores se resuelven de la manera siguiente después de utilizar la opción PREPROCESSOR:

```
EXEC SQL BEGIN DECLARE SECTION;
  char a[4];
  char b[12];
  struct
  {
    short length;
    char  data[18];
  } m;
  SQL TYPE IS BLOB(4) x;
  SQL TYPE IS CLOB(15) y;
  SQL TYPE IS DBCLOB(6144) z;
EXEC SQL END DECLARE SECTION;
```

Soporte de estructuras del lenguaje principal en C y C++

Con el soporte de estructuras del lenguaje principal, el precompilador C/C++ permite agrupar las variables del lenguaje principal en una sola estructura del

lenguaje principal. Esta función brinda una nota taquigráfica para hacer referencia a ese mismo conjunto de variables del lenguaje principal en una sentencia de SQL. Por ejemplo, se puede utilizar la siguiente estructura del lenguaje principal para acceder a algunas de las columnas de la tabla STAFF de la base de datos SAMPLE:

```
struct tag
   {
     short id;
     struct
     {
       short length;
       char  data[10];
     } name;
     struct
     {
       short   years;
       double salary;
     } info;
   } staff_record;
```

Los campos de una estructura del lenguaje principal pueden ser de cualquiera de los tipos de variable del lenguaje principal válidos. Los tipos válidos incluyen todos los tipos numéricos, de carácter y de objetos grande. También se soportan estructuras del lenguaje principal anidadas hasta 25 niveles. En el ejemplo anterior, el campo `info` es una subestructura, mientras que el campo `name` no lo es, puesto que representa un campo VARCHAR. Se aplica el mismo principio a LONG VARCHAR, VARGRAPHIC y LONG VARGRAPHIC. Asimismo, se soportan punteros a las estructuras del lenguaje principal.

Existen dos maneras de hacer referencia a las variables del lenguaje principal agrupadas en una estructura del lenguaje principal en una sentencia de SQL:

- Se puede hacer referencia al nombre de la estructura del lenguaje principal en una sentencia de SQL.

  ```
  EXEC SQL SELECT id, name, years, salary
       INTO :staff_record
       FROM staff
       WHERE id = 10;
  ```

  El precompilador convierte la referencia a `staff_record` en una lista de todos los campos declarados en la estructura del lenguaje principal, separados por comas. Cada campo se califica con los nombres de estructura del lenguaje principal de todos los niveles, a fin de evitar conflictos de denominación con otras variables del lenguaje principal o campos. Esto es equivalente al método siguiente.
- Se puede hacer referencia a nombres de estructuras del lenguaje principal completamente calificadas en una sentencia de SQL.

  ```
  EXEC SQL SELECT id, name, years, salary
       INTO :staff_record.id, :staff_record.name,
            :staff_record.info.years, :staff_record.info.salary
       FROM staff
       WHERE id = 10;
  ```

  Las referencias a nombres de campos se deben calificar por completo aunque no existan otras variables del lenguaje principal que tengan el mismo nombre. También se puede hacer referencia a subestructuras calificadas. En el ejemplo anterior, se puede utilizar `:staff_record.info` para sustituir a `:staff_record.info.years, :staff_record.info.salary`.

Puesto que una referencia a una estructura del lenguaje principal (primer ejemplo) es equivalente a una lista de sus campos, separados por comas, existen casos en que este tipo de referencia puede conducir a un error. Por ejemplo:

```
EXEC SQL DELETE FROM :staff_record;
```

Aquí, la sentencia DELETE espera una sola variable del lenguaje principal basada en caracteres. Si en su lugar se proporciona una estructura del lenguaje principal, la sentencia tiene como resultado un error durante la precompilación:

```
SQL0087N  La variable del lenguaje principal "staff_record" es una estructura
que se utiliza donde no se permiten referencias a estructuras.
```

Otros usos de las estructuras del lenguaje principal que pueden ocasionar que se produzca un error SQL0087N incluyen: PREPARE, EXECUTE IMMEDIATE, CALL, variables de indicador y referencias a SQLDA. Las estructuras del lenguaje principal que tengan exactamente un campo están permitidas en estas situaciones, puesto que constituyen referencias a campos individuales (segundo ejemplo).

Conceptos relacionados:

• "Tablas de indicadores en C y C++" en la página 168

Tablas de indicadores en C y C++

Una tabla de indicadores es un conjunto de variables de indicador que se utilizarán con una estructura del lenguaje principal. Se debe declarar como matriz de enteros cortos. Por ejemplo:

```
short ind_tab[10];
```

El ejemplo anterior declara una tabla de indicadores con 10 elementos. El siguiente ejemplo muestra la manera en que se puede utilizar en una sentencia de SQL:

```
EXEC SQL SELECT id, name, years, salary
     INTO :staff_record INDICATOR :ind_tab
     FROM staff
     WHERE id = 10;
```

A continuación se lista cada campo de la estructura del lenguaje principal con la variable de indicador correspondiente en la tabla:

staff_record.id	ind_tab[0]
staff_record.name	ind_tab[1]
staff_record.info.years	ind_tab[2]
staff_record.info.salary	ind_tab[3]

Nota: En una sentencia de SQL no se puede hacer referencia, de forma individual, a un elemento de una tabla de indicadores, por ejemplo `ind_tab[1]`. La palabra clave INDICATOR es opcional. No es necesario que el número de campos de estructura y de indicadores coincida; los indicadores de más no se utilizan ni tampoco los campos de más que no tienen indicadores asignados.

En lugar de una tabla de indicadores, también se puede utilizar una variable de indicador escalar para proporcionar un indicador para el primer campo de la estructura del lenguaje principal. Esto equivale a tener una tabla de indicadores con un solo elemento. Por ejemplo:

```
short scalar_ind;

EXEC SQL SELECT id, name, years, salary
          INTO :staff_record INDICATOR :scalar_ind
          FROM staff
          WHERE id = 10;
```

Si se especifica una tabla de indicadores junto con una variable del lenguaje principal en lugar de una estructura del lenguaje principal, sólo se utilizará el primer elemento de la tabla de indicadores, por ejemplo `ind_tab[0]`:

```
EXEC SQL SELECT id
          INTO :staff_record.id INDICATOR :ind_tab
          FROM staff
          WHERE id = 10;
```

Si se declara una matriz de enteros cortos en una estructura del lenguaje principal:

```
struct tag
{
  short i[2];
} test_record;
```

La matriz se expandirá en sus elementos cuando se haga referencia a `test_record` en una sentencia de SQL, haciendo que `:test_record` sea equivalente a `:test_record.i[0], :test_record.i[1]`.

Conceptos relacionados:

- "Soporte de estructuras del lenguaje principal en C y C++" en la página 166

Series terminadas en nulo en C y C++

Las series terminadas en nulo de C/C++ tienen su propio SQLTYPE (460/461 para caracteres y 468/469 para gráficos).

Las series terminadas en nulo de C/C++ se manejan de forma distinta en función del valor de la opción LANGLEVEL del precompilador. Si se especifica una variable del lenguaje principal con uno de estos valores SQLTYPE y la longitud declarada *n* dentro de una sentencia de SQL y el número de bytes de datos (para tipos de carácter) o de caracteres de doble byte (para tipos gráficos) es *k*, sucede lo siguiente:

- Si la opción LANGLEVEL del mandato PREP es SAA1 (valor por omisión):

  Para la salida:

Si...	Sucede que...
$k > n$	Se mueven *n* caracteres a la variable del lenguaje principal de destino, SQLWARN1 se establece en 'W' y SQLCODE es 0 (SQLSTATE 01004). No se coloca ningún terminador nulo en la serie. Si se ha especificado una variable de indicador con la variable del lenguaje principal, el valor de la variable de indicador se establece en *k*.
$k = n$	Se mueven *k* caracteres a la variable del lenguaje principal de destino, SQLWARN1 se establece en 'N' y SQLCODE es 0 (SQLSTATE 01004). No se coloca ningún terminador nulo en la serie. Si se ha especificado una variable de indicador con la

variable del lenguaje principal, el valor de la variable de indicador se establece en 0.

k < *n*	Se mueven *k* caracteres a la variable del lenguaje principal de destino y se coloca un carácter nulo en el carácter *k + 1*. Si se ha especificado una variable de indicador con la variable del lenguaje principal, el valor de la variable de indicador se establece en 0.

Para la entrada:

Cuando el gestor de bases de datos encuentre una variable del lenguaje principal de entrada que tiene uno de estos valores SQLTYPE y no finaliza por un terminador nulo, supondrá que el carácter *n+1* contendrá el carácter terminador nulo.

- Si la opción LANGLEVEL del mandato PREP es MIA:

  Para la salida:

Si...	Sucede que...
k >= *n*	Se mueven *n - 1* caracteres a la variable del lenguaje principal de destino, SQLWARN1 se establece en 'W' y SQLCODE es 0 (SQLSTATE 01004). El carácter número *n* se establece como terminador nulo. Si se ha especificado una variable de indicador con la variable del lenguaje principal, el valor de la variable de indicador se establece en *k*.
k + 1 = *n*	Se mueven *k* caracteres a la variable del lenguaje principal de destino y se coloca el terminador nulo en el carácter *n*. Si se ha especificado una variable de indicador con la variable del lenguaje principal, el valor de la variable de indicador se establece en 0.
k + 1 < *n*	Se mueven *k* caracteres a la variable del lenguaje principal de destino, se añaden *n - k -1* blancos a la derecha, a partir del carácter *k + 1*, y luego se coloca el terminador nulo en el carácter *n*. Si se ha especificado una variable de indicador con la variable del lenguaje principal, el valor de la variable de indicador se establece en 0.

  Para la entrada:

  Cuando el gestor de bases de datos encuentre una variable del lenguaje principal de entrada que tenga uno de estos valores SQLTYPE y no finalice por un carácter nulo, se devolverá SQLCODE -302 (SQLSTATE 22501).

Si se especifica en cualquier otro contexto de SQL, una variable del lenguaje principal con SQLTYPE 460 y longitud *n* se trata como el tipo de datos VARCHAR con longitud *n*, tal como se ha definido anteriormente. Si se especifica en cualquier otro contexto de SQL, una variable del lenguaje principal con SQLTYPE 468 y longitud *n* se trata como el tipo de datos VARGRAPHIC con longitud *n*, tal como se ha definido anteriormente.

Variables del lenguaje principal utilizadas como tipos de datos de puntero en C y C++

Las variables del lenguaje principal se pueden declarar como punteros a tipos de datos específicos, con las restricciones siguientes:

- Si una variable del lenguaje principal se declara como puntero, no se puede declarar ninguna otra variable del lenguaje principal con el mismo nombre dentro del mismo archivo fuente. El ejemplo siguiente no está permitido:

```
char mystring[20];
char (*mystring)[20];
```

- Cuando declare un puntero a una matriz de caracteres terminada en nulo, utilice paréntesis. En todos los casos restantes, no se permiten paréntesis. Por ejemplo:

```
EXEC SQL BEGIN DECLARE SECTION;
   char  (*arr)[10];  /* correcto  */
   char  *(arr);      /* incorrecto */
   char  *arr[10];    /* incorrecto */
 EXEC SQL END DECLARE SECTION;
```

  La primera declaración es un puntero a una matriz de caracteres de 10 bytes. Esta variable del lenguaje principal es válida. La segunda declaración no es válida. Los paréntesis no están permitidos en un puntero a un carácter. La tercera declaración es una matriz de punteros. Este tipo de datos no se soporta.

  La declaración de variable del lenguaje principal:

```
char *ptr
```

  se acepta, pero no significa *serie de caracteres terminada en nulo de longitud indeterminada*; significa *puntero a una variable del lenguaje principal de un solo carácter y de longitud fija*. Es posible que esto no fuera lo que se pretendía. Para definir una variable del lenguaje principal de puntero que pueda indicar distintas series de caracteres, utilice el primero de los formatos de declaración anteriores.

- Cuando se utilizan variables del lenguaje principal de puntero en sentencias de SQL, deben tener como prefijo el mismo número de asteriscos con que se han declarado, tal como en el ejemplo siguiente:

```
EXEC SQL BEGIN DECLARE SECTION;
  char (*mychar)[20];    /* Puntero a matriz de caracteres de 20 bytes */
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT column INTO :*mychar FROM table;   /* Correcta */
```

- Sólo se puede utilizar el asterisco como operador en un nombre de variable del lenguaje principal.
- La longitud máxima de un nombre de variable del lenguaje principal no se ve afectada por el número de asteriscos especificados, puesto que no se considera que los asteriscos formen parte del nombre.
- Siempre que utilice una variable de puntero en una sentencia de SQL, debe dejar la opción de precompilación del nivel de optimización (OPTLEVEL) en su valor por omisión, que es 0 (sin optimización). Esto significa que el gestor de bases de datos no realizará ninguna optimización de la SQLDA.

Miembros de datos de clase utilizados como variables del lenguaje principal en C y C++

Puede declarar miembros de datos de clase como variables del lenguaje principal (pero no clases u objetos en sí mismos). El ejemplo siguiente ilustra el método a utilizar:

```
class STAFF
{
    private:
         EXEC SQL BEGIN DECLARE SECTION;
           char        staff_name[20];
           short int   staff_id;
           double      staff_salary;
         EXEC SQL END DECLARE SECTION;
         short       staff_in_db;
    .
    .
};
```

A los miembros de datos sólo se puede acceder directamente en sentencias de SQL mediante el puntero *this* implícito proporcionado por el compilador C++ en las funciones de miembros de clases. **No se puede** calificar explícitamente una instancia de un objeto (como, por ejemplo, `SELECT name INTO :my_obj.staff_name ...`) en una sentencia de SQL.

Si se hace una referencia directa a miembros de datos de clase en sentencias de SQL, el gestor de bases de datos resuelve la referencia utilizando el puntero *this*. Por este motivo, debe dejar la opción de precompilación del nivel de optimización (OPTLEVEL) en su valor por omisión, que es 0 (sin optimización). Esto significa que el gestor de bases de datos no realizará ninguna optimización de la SQLDA. (Esto es así siempre que existan variables del lenguaje principal de puntero implicadas en las sentencias de SQL.)

El ejemplo siguiente muestra cómo se pueden utilizar directamente los miembros de datos de clase que ha declarado como variables del lenguaje principal en una sentencia de SQL.

```
class STAFF
{

⋮
    public:


⋮
       short int hire( void )
       {
         EXEC SQL INSERT INTO staff ( name,id,salary )
           VALUES ( :staff_name, :staff_id, :staff_salary );
         staff_in_db = (sqlca.sqlcode == 0);
         return sqlca.sqlcode;
       }
};
```

En este ejemplo, los miembros de datos de clase `staff_name`, `staff_id` y `staff_salary` se utilizan directamente en la sentencia INSERT. Puesto que se han declarado como variables del lenguaje principal (consulte el primer ejemplo de esta sección), están calificados de forma implícita para el objeto actual con el puntero *this*. En sentencias de SQL también se puede hacer referencia a miembros de datos a los que no se pueda acceder mediante el puntero *this*. Esto se logra haciendo una referencia indirecta a ellos mediante variables del lenguaje principal de puntero o de referencia.

El ejemplo siguiente muestra un nuevo método, *asWellPaidAs*, que toma un segundo objeto, *otherGuy*. Este método hace referencia a sus miembros

indirectamente, mediante una variable del lenguaje principal de puntero local o de referencia, puesto que no se puede hacer una referencia directa a sus miembros dentro de la sentencia de SQL.

```
short int STAFF::asWellPaidAs( STAFF otherGuy )
{
    EXEC SQL BEGIN DECLARE SECTION;
      short &otherID = otherGuy.staff_id
      double otherSalary;
    EXEC SQL END DECLARE SECTION;
    EXEC SQL SELECT SALARY INTO :otherSalary
      FROM STAFF WHERE id = :otherID;
      if( sqlca.sqlcode == 0 )
         return staff_salary >= otherSalary;
      else
         return 0;
}
```

Operadores de calificación y de miembro en C y C++

No se puede utilizar el operador de resolución de ámbito de C++ '::' ni los operadores de miembros de C/C++ '.' ni '->' en sentencias de SQL incorporado. Se puede lograr fácilmente lo mismo mediante el uso de variables de puntero o de referencia locales, que se establecen fuera de la sentencia de SQL, de forma que apunten a la variable de ámbito que se desee y luego se utilizan dentro de la sentencia de SQL para hacer referencia a ésta. El ejemplo siguiente muestra el método correcto a utilizar:

```
EXEC SQL BEGIN DECLARE SECTION;
  char (& localName)[20] = ::name;
EXEC SQL END DECLARE SECTION;
EXEC SQL
  SELECT name INTO :localName FROM STAFF
  WHERE name = 'Sanders';
```

Codificación de caracteres de varios bytes en C y C++

Algunos esquemas de codificación de caracteres, especialmente aquéllos que proceden de los países del este asiático, necesitan varios bytes para representar un carácter. Esta representación externa de los datos se denomina representación de un carácter por medio de *código de caracteres de varios bytes* e incluye los caracteres de doble byte (caracteres representados por dos bytes). Los datos gráficos en DB2® consisten en caracteres de doble byte.

Para manipular series de caracteres que contienen caracteres de doble byte puede resultar conveniente que una aplicación utilice una representación interna de los datos. Esta representación interna se denomina representación de caracteres de doble byte por medio de *código de caracteres amplios* y es el formato utilizado habitualmente en el tipo de datos `wchar_t` de C/C++. Se dispone de subrutinas que se ajustan a C de ANSI y a X/OPEN Portability Guide 4 (XPG4) para procesar los datos de caracteres amplios y para convertir los datos que se encuentran en este formato al formato de varios bytes, y viceversa.

Observe que, aunque una aplicación puede procesar datos de tipo carácter en formato de varios bytes o en formato de caracteres amplios, la interacción con el gestor de bases de datos sólo se realiza con códigos de caracteres DBCS (de varios bytes). Es decir, los datos se almacenan y se recuperan de las columnas GRAPHIC en formato DBCS. Se proporciona la opción WCHARTYPE del precompilador para permitir que los datos de una aplicación que están en formato de caracteres

amplios se conviertan al formato de varios bytes, y viceversa, cuando se produce el intercambio de datos con el motor de la base de datos.

Conceptos relacionados:

- "Variables gráficas del lenguaje principal en C y C++" en la página 158
- "Tipos de datos wchar_t y sqldbchar en C y C++" en la página 174

Tipos de datos wchar_t y sqldbchar en C y C++

Mientras que el tamaño y la codificación de datos gráficos de DB2® son constantes en todas las plataformas para una página de códigos determinada, el tamaño y el formato interno del tipo de datos `wchar_t` de C o de C++ de ANSI dependen del compilador que se utilice y de la plataforma en que se encuentre. No obstante, DB2 define el tipo de datos `sqldbchar` con un tamaño de dos bytes, y se pretende que constituya una manera portátil de manipular los datos DBCS y UCS-2 en el mismo formato en que están almacenados en la base de datos.

Puede definir todos los tipos de variables del lenguaje principal de gráficos C de DB2 mediante `wchar_t` o `sqldbchar`. Debe utilizar `wchar_t` si crea la aplicación mediante la opción de precompilación WCHARTYPE CONVERT.

Nota: Cuando especifique la opción WCHARTYPE CONVERT en una plataforma Windows®, tenga en cuenta que `wchar_t` en plataformas Windows es Unicode. Por lo tanto, si el `wchar_t` del compilador C/C++ no es Unicode, es posible que la llamada a la función `wcstombs()` falle con SQLCODE -1421 (SQLSTATE=22504). Si esto sucede, puede especificar la opción WCHARTYPE NOCONVERT y llamar a las funciones `wcstombs()` y `mbstowcs()` de forma explícita desde dentro del programa.

Si crea la aplicación con la opción de precompilación WCHARTYPE NOCONVERT, debe utilizar `sqldbchar` para conseguir una portabilidad máxima entre distintas plataformas de cliente y servidor DB2. Puede utilizar `wchar_t` con WCHARTYPE NOCONVERT, pero sólo en aquellas plataformas en que `wchar_t` esté definido con una longitud de dos bytes.

Si utiliza incorrectamente `wchar_t` o `sqldbchar` en declaraciones de variables del lenguaje principal, durante la precompilación recibirá un SQLCODE 15 (sin SQLSTATE).

Conceptos relacionados:

- "Opción del precompilador WCHARTYPE en C y C++" en la página 174
- "Consideraciones sobre los juegos de códigos EUC y UCS-2 de japonés y chino tradicional" en la página 656

Opción del precompilador WCHARTYPE en C y C++

Mediante la opción WCHARTYPE del precompilador, puede especificar el formato de caracteres gráficos que desea utilizar en la aplicación C/C++. Esta opción proporciona flexibilidad para elegir entre tener los datos gráficos en formato de varios bytes o en formato de caracteres amplios. La opción WCHARTYPE tiene dos valores posibles:

CONVERT
Si se selecciona la opción WCHARTYPE CONVERT, se convierten los códigos de caracteres entre la variable de gráficos del lenguaje principal y

el gestor de bases de datos. Para las variables del lenguaje principal de entrada de gráficos, la conversión de código de caracteres del formato de caracteres amplios a formato de caracteres DBCS de varios bytes se realiza antes de enviar los datos al gestor de bases de datos, mediante la función `wcstombs()` de C de ANSI. Para las variables del lenguaje principal de salida de gráficos, la conversión de código de caracteres del formato de caracteres DBCS de varios bytes al formato de caracteres amplios se realiza antes de que los datos recibidos del gestor de bases de datos se almacenen en la variable del lenguaje principal, mediante la función `mbstowcs()` de C de ANSI.

La ventaja de utilizar WCHARTYPE CONVERT es que permite que la aplicación aproveche plenamente los mecanismos de C de ANSI para tratar las series de caracteres amplios (literales L, funciones de serie 'wc', etc.) sin tener que convertir los datos de forma explícita al formato de varios bytes antes de establecer comunicación con el gestor de bases de datos. El inconveniente es que las conversiones implícitas puede tener un impacto sobre el rendimiento de la aplicación durante la ejecución de ésta y pueden incrementar los requisitos de memoria.

Si selecciona WCHARTYPE CONVERT, declare todas las variables del lenguaje principal de gráficos mediante `wchar_t` en lugar de mediante `sqldbchar`.

Si desea el comportamiento de WCHARTYPE CONVERT pero no es necesario precompilar la aplicación (por ejemplo, una aplicación CLI), defina la macro `SQL_WCHART_CONVERT` del preprocesador de C durante la compilación. Así se asegurará de que determinadas definiciones de los archivos de cabecera de DB2 utilicen el tipo de datos `wchar_t` en lugar de `sqldbchar`.

Nota: La opción de precompilación WCHARTYPE CONVERT no recibe soporte actualmente en programas que se ejecutan en el cliente DB2® Windows® 3.1. Para estos programas, utilice el valor por omisión (WCHARTYPE NOCONVERT).

NOCONVERT (valor por omisión)
Si se elige la opción WCHARTYPE NOCONVERT o no se especifica ninguna opción WCHARTYPE, no se produce ninguna conversión implícita del código de caracteres entre la aplicación y el gestor de bases de datos. Los datos de una variable del lenguaje principal de gráficos se envían al gestor de bases de datos y se reciben del mismo como caracteres DBCS inalterados. Esto tiene la ventaja de que se mejora el rendimiento, pero el inconveniente de que la aplicación debe abstenerse de utilizar datos de caracteres amplios en las variables del lenguaje principal `wchar_t`, o tiene que llamar explícitamente a las funciones `wcstombs()` y `mbstowcs()` para convertir los datos al formato de varios bytes, o viceversa, cuando interactúe con el gestor de bases de datos.

Si selecciona WCHARTYPE NOCONVERT, declare todas las variables del lenguaje principal de gráficos utilizando el tipo `sqldbchar`, a fin de obtener la máxima portabilidad a otras plataformas cliente/servidor de DB2.

Hay otras directrices que debe tener en cuenta, que son las siguientes:

- Puesto que se utiliza el soporte de `wchar_t` o `sqldbchar` para manejar datos DBCS, su uso requiere que el hardware y el software tengan capacidad de DBCS o EUC. Este soporte sólo está disponible en el entorno DBCS de DB2 Universal

Database, o para tratar datos GRAPHIC en cualquier aplicación (incluidas las aplicaciones de un solo byte) conectada a una base de datos UCS-2.

- Los caracteres que no sean DBCS, y los caracteres amplios que se pueden convertir en caracteres no DBCS, no se deben utilizar en series gráficas. *Caracteres que no sean DBCS* hace referencia a los caracteres de un solo byte y a los caracteres que no son de doble byte. Las series gráficas no se validan para asegurar que sus valores únicamente contienen puntos de código de caracteres de doble byte. Las variables del lenguaje principal de gráficos sólo deben contener datos DBCS o, si WCHARTYPE CONVERT está en vigor, datos de caracteres amplios que se convierten en datos DBCS. Debe almacenar los datos que contienen una mezcla de caracteres de doble byte y de un solo byte en variables del lenguaje principal de tipo carácter. Observe que las variables del lenguaje principal de datos mixtos no se ven afectadas por el establecimiento de la opción WCHARTYPE.
- En las aplicaciones en que se utiliza la opción de precompilación WCHARTYPE NOCONVERT, no se deben utilizar literales L junto con variables del lenguaje principal de gráficos, puesto que los literales L están en formato de caracteres amplios. Un literal L es una serie de caracteres amplios C que tiene como prefijo la letra L y como tipo de datos `"array of wchar_t"`. Por ejemplo, `L"dbcs-string"` es un literal L.
- En las aplicaciones en que se utiliza la opción de precompilación WCHARTYPE CONVERT, se pueden utilizar literales L para inicializar variables del lenguaje principal `wchar_t`, pero no se pueden utilizar en sentencias de SQL. En lugar de utilizar literales L, las sentencias de SQL deben usar constantes de serie de gráficos, que son independientes del valor de WCHARTYPE.
- El valor de la opción WCHARTYPE afecta a los datos de gráficos que se pasan al gestor de bases de datos y que se reciben de éste utilizando la estructura SQLDA, así como a las variables del lenguaje principal. Si WCHARTYPE CONVERT está en vigor, se supondrá que los datos gráficos recibidos de la aplicación mediante una SQLDA están en formato de caracteres amplios, y se convertirán al formato DBCS a través de una llamada implícita a `wcstombs()`. De forma parecida, los datos de salida de gráficos recibidos por una aplicación se habrán convertido al formato de caracteres amplios antes de colocarlos en el almacenamiento de la aplicación.
- Los procedimientos almacenados sin barrera se deben precompilar con la opción WCHARTYPE NOCONVERT. Los procedimientos almacenados normales con barrera se pueden precompilar con las opciones CONVERT o NOCONVERT, lo cual afectará al formato de los datos gráficos manipulados por las sentencias de SQL contenidas en el procedimiento almacenado. No obstante, y en cualquier caso, los datos gráficos que se pasen al procedimiento almacenado mediante la SQLDA estarán en formato DBCS. De la misma forma, los datos que se pasen desde el procedimiento almacenado mediante la SQLDA deben estar en formato DBCS.
- Si una aplicación llama a un procedimiento almacenado a través de la interfaz Database Application Remote Interface (DARI) (la API `sqleproc()`), los datos gráficos de la SQLDA de entrada tienen que estar en formato DBCS, o en UCS-2 si está conectada a una base de datos UCS-2, independientemente del estado del valor de WCHARTYPE de la aplicación que realiza la llamada. Asimismo, los datos gráficos de la SQLDA de salida se devolverán en formato DBCS, o en UCS-2 si está conectada a una base de datos UCS-2, independientemente del valor de WCHARTYPE.
- Si una aplicación llama a un procedimiento almacenado mediante la sentencia CALL de SQL, se producirá una conversión de los datos gráficos en la SQLDA, según el valor de WCHARTYPE de la aplicación que realiza la llamada.

- Los datos gráficos que se pasen a funciones definidas por el usuario (UDF) siempre estarán en formato DBCS. De la misma forma, se supondrá que los datos gráficos devueltos desde una UDF están en formato DBCS para las bases de datos DBCS, y en formato UCS-2 para las bases de datos EUC y UCS-2.
- Los datos almacenados en archivos DBCLOB mediante el uso de variables de referencia a archivos DBCLOB se almacenan en formato DBCS o, en el caso de bases de datos UCS-2, en formato UCS-2. Del mismo modo, los datos de entrada procedentes de archivos DBCLOB se recuperan en formato DBCS o, en el caso de bases de datos UCS-2, en formato UCS-2.

Nota: Si se precompilan aplicaciones C utilizando la opción WCHARTYPE CONVERT, DB2 valida los datos gráficos de la aplicación tanto de entrada como de salida, puesto que los datos se pasan a través de las funciones de conversión. Si *no* se utiliza la opción CONVERT, no se produce ninguna conversión de los datos gráficos, por lo que tampoco se produce ninguna validación. En un entorno mixto de CONVERT/NOCONVERT, se pueden ocasionar problemas si una aplicación NOCONVERT inserta datos gráficos no válidos que luego son captados por una aplicación CONVERT. Estos datos no se pueden convertir con SQLCODE -1421 (SQLSTATE 22504) en una operación FETCH de la aplicación CONVERT.

Información relacionada:

- "Sentencia PREPARE" en la publicación *Consulta de SQL, Volumen 2*

Consideraciones sobre EUC en japonés o chino tradicional y UCS-2 en C y C++

Si la página de códigos de la aplicación es EUC en japonés o chino tradicional, o si la aplicación conecta con una base de datos UCS-2, puede acceder a columnas GRAPHIC del servidor de bases de datos utilizando las opciones CONVERT o NOCONVERT y las variables del lenguaje principal de gráficos `wchar_t` o `sqldbchar`, o las SQLDA de entrada/salida. En este apartado, *formato DBCS* hace referencia al esquema de codificación de UCS-2 para datos EUC. Considere los casos siguientes:

- Se utiliza la opción CONVERT

  El cliente DB2® convierte los datos gráficos del formato de caracteres amplios a la página de códigos de la aplicación, y luego a UCS-2, antes de enviar la SQLDA de entrada al servidor de bases de datos. Los datos gráficos se envían al servidor de bases de datos identificados por el identificador de página de códigos de UCS-2. Los datos de tipo carácter mixtos se identifican con el identificador de página de códigos de la aplicación. Cuando un cliente recupera datos gráficos de una base de datos, éstos se identifican con el identificador de página de códigos de UCS-2. El cliente DB2 convierte los datos de UCS-2 a la página de códigos de la aplicación cliente y luego al formato de caracteres amplios. Si se utiliza una SQLDA de entrada en lugar de una variable del lenguaje principal, se requiere al usuario que se asegure de que los datos gráficos se han codificado utilizando el formato de caracteres amplios. Estos datos se convertirán a UCS-2 y luego se enviarán al servidor de bases de datos. Estas conversiones tendrán impacto en el rendimiento.
- Se utiliza la opción NOCONVERT

  DB2 supone que los datos se han codificado utilizando UCS-2 y están identificados por la página de códigos de UCS-2, y no se produce ninguna conversión. DB2 supone que la variable del lenguaje principal de gráficos se utiliza simplemente como contenedor. Si no se selecciona la opción

NOCONVERT, los datos gráficos recuperados del servidor de bases de datos se pasan a la aplicación codificados mediante UCS-2. Las posibles conversiones de la página de códigos de la aplicación a UCS-2 y de UCS-2 a la página de códigos de la aplicación son responsabilidad del usuario. Los datos identificados como UCS-2 se envían al servidor de bases de datos sin que se produzca ninguna conversión ni alteración.

Para minimizar las conversiones, puede utilizar la opción NOCONVERT y manejar las conversiones en la aplicación, o bien no utilizar columnas GRAPHIC. Para los entornos de cliente en que la codificación `wchar_t` es en Unicode de dos bytes, por ejemplo en Windows® NT o AIX® versión 4.3 y posteriores, puede utilizar la opción NOCONVERT y trabajar directamente con UCS-2. En estos casos, la aplicación debe manejar la diferencia entre las arquitecturas big-endian y little-endian. Con la opción NOCONVERT, DB2 Universal Database utiliza `sqldbchar`, que es siempre un big-endian de dos bytes.

No asigne datos IBM® CS0 (ASCII de 7 bits) e IBM-eucJP CS2 (Katakana) a variables gráficas de lenguaje principal después de una conversión a UCS-2 (si se especifica NOCONVERT) ni mediante una conversión al formato de caracteres amplios (si se especifica CONVERT). Esto es así porque los caracteres de estos juegos de códigos EUC pasan a tener un solo byte cuando se convierten de UCS-2 a DBCS de PC.

En general, aunque eucJP y eucTW almacenan los datos gráficos (GRAPHIC) como UCS-2, los datos GRAPHIC contenidos en estas bases de datos siguen siendo datos eucJP o eucTW no ASCII. Concretamente, cualquier espacio rellenado en estos datos GRAPHIC es espacio DBCS (también conocido como espacio ideográfico en UCS-2, U+3000). Sin embargo, para una base de datos UCS-2, los datos GRAPHIC pueden contener cualquier carácter UCS-2 y el relleno del espacio se realiza con espacio UCS-2, U+0020. Tenga presente esta diferencia cuando codifique aplicaciones para recuperar datos UCS-2 de una base de datos UCS-2, en comparación con la recuperación de datos UCS-2 de bases de datos eucJP y eucTW.

Conceptos relacionados:

- "Consideraciones sobre los juegos de códigos EUC y UCS-2 de japonés y chino tradicional" en la página 656

Sección declare de SQL con variables del lenguaje principal para C y C++

A continuación se muestra una sección de declaración de SQL de ejemplo con variables del lenguaje principal declaradas para tipos de datos SQL soportados:

```
EXEC SQL BEGIN DECLARE SECTION;

:
        short     age = 26;              /* tipo de SQL  500 */
        short     year;                  /* tipo de SQL  500 */
        sqlint32  salary;                /* tipo de SQL  496 */
        sqlint32  deptno;                /* tipo de SQL  496 */
        float     bonus;                 /* tipo de SQL  480 */
        double    wage;                  /* tipo de SQL  480 */
        char      mi;                    /* tipo de SQL  452 */
        char      name[6];               /* tipo de SQL  460 */
        struct   {
                  short len;
                  char data[24];
```

```
          } address;               /* tipo de SQL  448 */
        struct   {
                  short len;
                  char data[32695];
                 } voice;                  /* tipo de SQL  456 */
        sql type is clob(1m)
                  chapter;                 /* tipo de SQL  408 */
        sql type is clob_locator
                  chapter_locator;         /* tipo de SQL  964 */
        sql type is clob_file
                  chapter_file_ref;        /* tipo de SQL  920 */
        sql type is blob(1m)
                  video;                   /* tipo de SQL  404 */
        sql type is blob_locator
                  video_locator;           /* tipo de SQL  960 */
        sql type is blob_file
                  video_file_ref;          /* tipo de SQL  916 */
        sql type is dbclob(1m)
                  tokyo_phone_dir;         /* tipo de SQL  412 */
        sql type is dbclob_locator
                  tokyo_phone_dir_lctr;    /* tipo de SQL  968 */
        sql type is dbclob_file
                  tokyo_phone_dir_flref;   /* tipo de SQL  924 */
        struct   {
                  short len;
                  sqldbchar data[100];
                 } vargraphic1;           /* tipo de SQL  464 */
                                          /* Precompilado con la
                                          opción WCHARTYPE NOCONVERT */
        struct   {
                  short len;
                  wchar_t data[100];
                 } vargraphic2;           /* tipo de SQL  464 */
                                          /* Precompilado con la
                                          opción WCHARTYPE CONVERT */
        struct   {
                  short len;
                  sqldbchar data[10000];
                 } long_vargraphic1;      /* tipo de SQL  472 */
                                          /* Precompilado con la
                                          opción WCHARTYPE NOCONVERT */
        struct   {
                  short len;
                  wchar_t data[10000];
                 } long_vargraphic2;      /* tipo de SQL  472 */
                                          /* Precompilado con la
                                          opción WCHARTYPE CONVERT */
        sqldbchar graphic1[100];          /* tipo de SQL  468 */
                                          /* Precompilado con la
                                          opción WCHARTYPE NOCONVERT */
        wchar_t   graphic2[100];          /* tipo de SQL  468 */
                                          /* Precompilado con la
                                          opción WCHARTYPE CONVERT */
        char      date[11];               /* tipo de SQL  384 */
        char      time[9];                /* tipo de SQL  388 */
        char      timestamp[27];          /* tipo de SQL  392 */
        short     wage_ind;               /* Indicador de nulo */

    .
    .
    .
      EXEC SQL END DECLARE SECTION;
```

Consideraciones sobre tipos de datos para C y C++

Las secciones siguientes describen cómo se correlacionan los tipos de datos de SQL con tipos de datos de C y C++.

Tipos de datos SQL soportados en C y C++

Ciertos tipos de datos predefinidos de C y C++ corresponden con los tipos de columna del gestor de bases de datos. Sólo se pueden declarar como variables del lenguaje principal estos tipos de datos de C/C++.

La tabla siguiente muestra el equivalente en C/C++ de cada tipo de columna. Cuando el precompilador encuentra una declaración de una variable del lenguaje principal, determina el valor del tipo de SQL apropiado. El gestor de bases de datos utiliza este valor para convertir los datos intercambiados entre la aplicación y él mismo.

Nota: No existe soporte de variables del lenguaje principal para el tipo de datos DATALINK en ninguno de los lenguajes principales de DB2.

Tabla 13. Tipos de datos SQL correlacionados con declaraciones C/C++

Tipo de columna SQL[1]	Tipo de datos C/C++	Descripción del tipo de columna SQL
SMALLINT (500 ó 501)	short short int sqlint16	Entero con signo de 16 bits
INTEGER (496 ó 497)	long long int sqlint32[2]	Entero con signo de 32 bits
BIGINT (492 ó 493)	long long long __int64 sqlint64[3]	Entero con signo de 64 bits
REAL[4] (480 ó 481)	float	Coma flotante de precisión simple
DOUBLE[5] (480 ó 481)	double	Coma flotante de precisión doble
DECIMAL(*p*,*s*) (484 ó 485)	No existe un equivalente exacto; utilice double	Decimal empaquetado (Considere la posibilidad de utilizar las funciones CHAR y DECIMAL para manipular los campos decimales empaquetados como datos de tipo carácter.)
CHAR(1)(452 ó 453)	char	Carácter simple
CHAR(*n*) (452 ó 453)	No existe un equivalente exacto; utilice char[*n+1*], donde n es suficientemente grande para contener los datos 1<=*n*<=254	Serie de caracteres de longitud fija

Tabla 13. Tipos de datos SQL correlacionados con declaraciones C/C++ (continuación)

Tipo de columna SQL[1]	Tipo de datos C/C++	Descripción del tipo de columna SQL

VARCHAR(*n*)
(448 ó 449)
	struct tag {
	short int;
	char[*n*]
	}
	
	1<=*n*<=32 672
		Serie de caracteres de longitud variable no terminada en nulo con indicador de longitud de serie de 2 bytes

	Como alternativa, utilice char[*n+1*], donde n es suficientemente grande para contener los datos 1<=*n*<=32 672	Serie de caracteres de longitud variable terminada en nulo **Nota:** Se le asigna un tipo SQL de 460/461.

LONG VARCHAR
(456 ó 457)
	struct tag {
	short int;
	char[*n*]
	}
	
	32 673<=*n*<=32 700
		Serie de caracteres de longitud variable no terminada en nulo con indicador de longitud de serie de 2 bytes

CLOB(*n*) (408 ó 409)	sql type is clob(*n*) 1<=*n*<=2 147 483 647	Serie de caracteres de longitud variable no terminada en nulo con indicador de longitud de serie de 4 bytes
Variable de localizador CLOB[6] (964 ó 965)	sql type is clob_locator	Identifica las entidades CLOB que residen en el servidor
Variable de referencia de archivo CLOB[6] (920 ó 921)	sql type is clob_file	Descriptor del archivo que contiene datos CLOB
BLOB(*n*) (404 ó 405)	sql type is blob(*n*) 1<=*n*<=2 147 483 647	Serie binaria variable no terminada en nulo con indicador de longitud de serie de 4 bytes
Variable de localizador BLOB[6] (960 ó 961)	sql type is blob_locator	Identifica las entidades BLOB del servidor
Variable de referencia de archivo BLOB[6] (916 ó 917)	sql type is blob_file	Descriptor del archivo que contiene datos BLOB
DATE (384 ó 385)	Formato de caracteres terminado en nulo	Admitir como mínimo 11 caracteres para acomodar el terminador nulo.
	Formato estructurado VARCHAR	Admitir como mínimo 10 caracteres.
TIME (388 ó 389)	Formato de caracteres terminado en nulo	Admitir como mínimo 9 caracteres para acomodar el terminador nulo.
	Formato estructurado VARCHAR	Admitir como mínimo 8 caracteres.
TIMESTAMP (392 ó 393)	Formato de caracteres terminado en nulo	Admitir como mínimo 27 caracteres para acomodar el terminador nulo.
	Formato estructurado VARCHAR	Admitir como mínimo 26 caracteres.

Nota: Los tipos de datos siguientes sólo están disponibles en los entornos DBCS o EUC si se compilan previamente con la opción WCHARTYPE NOCONVERT.

Tabla 13. Tipos de datos SQL correlacionados con declaraciones C/C++ (continuación)

Tipo de columna SQL[1]	Tipo de datos C/C++	Descripción del tipo de columna SQL
GRAPHIC(1) (468 ó 469)	sqldbchar	Carácter simple de doble byte
GRAPHIC(*n*) (468 ó 469)	No existe un equivalente exacto; utilice sqldbchar[*n+1*], donde n es suficientemente grande para contener los datos 1<=*n*<=127	Serie de caracteres de doble byte y longitud fija

VARGRAPHIC(*n*)
(464 ó 465)
	struct tag {
	short int;
	sqldbchar[*n*]
	}
	
	1<=*n*<=16 336
		Serie de caracteres de doble byte y longitud variable no terminada en nulo con indicador de longitud de serie de 2 bytes

	Como alternativa, utilice sqldbchar[*n+1*], donde n es suficientemente grande para contener los datos 1<=*n*<=16 336	Serie de caracteres de doble byte y longitud variable terminada en nulo **Nota:** Se le asigna un tipo SQL de 400/401.

LONG VARGRAPHIC
(472 ó 473)
	struct tag {
	short int;
	sqldbchar[*n*]
	}
	
	16 337<=*n*<=16 350
		Serie de caracteres de doble byte y longitud variable no terminada en nulo con indicador de longitud de serie de 2 bytes

Nota: Los tipos de datos siguientes sólo están disponibles en los entornos DBCS o EUC si se compilan previamente con la opción WCHARTYPE CONVERT.		
GRAPHIC(1) (468 ó 469)	wchar_t	• Carácter amplio simple (para tipo C) • Carácter de doble byte simple (para tipo de columna)
GRAPHIC(*n*) (468 ó 469)	No existe un equivalente exacto; utilice wchar_t [*n+1*], donde n es suficientemente grande para contener los datos 1<=*n*<=127	Serie de caracteres de doble byte y longitud fija

VARGRAPHIC(*n*)
(464 ó 465)
	struct tag {
	short int;
	wchar_t [*n*]
	}
	
	1<=*n*<=16 336
		Serie de caracteres de doble byte y longitud variable no terminada en nulo con indicador de longitud de serie de 2 bytes

	Como alternativa, utilice char[*n+1*], donde n es suficientemente grande para contener los datos 1<=*n*<=16 336	Serie de caracteres de doble byte y longitud variable terminada en nulo **Nota:** Se le asigna un tipo SQL de 400/401.

Tabla 13. Tipos de datos SQL correlacionados con declaraciones C/C++ (continuación)

Tipo de columna SQL[1]	Tipo de datos C/C++	Descripción del tipo de columna SQL

LONG VARGRAPHIC
(472 ó 473)
	struct tag {
	short int;
	wchar_t [*n*]
	}
	
	16 337<=*n*<=16 350
		Serie de caracteres de doble byte y longitud variable no terminada en nulo con indicador de longitud de serie de 2 bytes

Nota: Los tipos de datos siguientes sólo están disponibles en los entornos DBCS o EUC.		
DBCLOB(*n*) (412 ó 413)	sql type is dbclob(*n*) 1<=*n*<=1 073 741 823	Serie de caracteres de doble byte y longitud variable no terminada en nulo con indicador de longitud de serie de 4 bytes
Variable de localizador DBCLOB[6] (968 ó 969)	sql type is dbclob_locator	Identifica las entidades DBCLOB que residen en el servidor
Variable de referencia de archivos DBCLOB[6] (924 ó 925)	sql type is dbclob_file	Descriptor del archivo que contiene datos DBCLOB

Notas:

1. El primer número que se encuentra bajo **Tipo de columna SQL** indica que no se proporciona una variable de indicador, y el segundo número indica que se proporciona una variable de indicador. Se necesita una variable de indicador para indicar los valores nulos (NULL) o para contener la longitud de una serie truncada. Éstos son los valores que aparecerán en el campo SQLTYPE de la SQLDA para estos tipos de datos.
2. Por razones de compatibilidad entre plataformas, utilice sqlint32. En las plataformas UNIX de 64 bits, "long" es un entero de 64 bits. En los sistemas operativos Windows de 64 bits y en las plataformas UNIX de 32 bits, "long" es un entero de 32 bits.
3. Por razones de compatibilidad entre plataformas, utilice sqlint64. El archivo de cabecera sqlsystm.h de DB2 Universal Database definirá el tipo sqlint64 como "__int64" en la plataforma Windows NT cuando se utilice el compilador de Microsoft, como "long long" en las plataformas UNIX de 32 bits y como "long" en las plataformas UNIX de 64 bits.
4. FLOAT(*n*) donde 0 < *n* < 25 es un sinónimo de REAL. La diferencia entre REAL y DOUBLE en la SQLDA es el valor de la longitud (4 u 8).
5. Los tipos SQL siguientes son sinónimos de DOUBLE:
   - FLOAT
   - FLOAT(*n*) donde 24 < *n* < 54 es
   - DOUBLE PRECISION
6. Éste no es un tipo de columna, sino un tipo de variable del lenguaje principal.

A continuación mostramos normas adicionales para los tipos de datos C/C++ soportados:

- El tipo de datos `char` se puede declarar como `char` o `unsigned char`.
- El gestor de bases de datos procesa los datos de serie de caracteres de longitud variable y terminados en nulo del tipo char[*n*] (tipo de datos 460) como VARCHAR(*m*).
  - Si LANGLEVEL es SAA1, la longitud de la variable del lenguaje principal *m* es igual a la longitud de la serie de caracteres *n* en char[*n*] o al número de bytes que preceden al primer terminador nulo (\0), el valor más pequeño de ambos.
  - Si LANGLEVEL es MIA, la longitud de la variable del lenguaje principal *m* es igual al número de bytes que preceden al primer terminador nulo (\0).

- El gestor de bases de datos procesa el tipo de datos de serie de caracteres de gráficos y longitud variable terminados en nulo, `wchar_t[n]` o `sqldbchar[n]` (tipo de datos 400), como VARGRAPHIC(*m*).
  - Si LANGLEVEL es SAA1, la longitud de la variable del lenguaje principal *m* es igual a la longitud de la serie de caracteres *n* en `wchar_t[n]` o `sqldbchar[n]`, o al número de caracteres que preceden al primer terminador nulo de gráficos, el valor más pequeño de ambos.
  - Si LANGLEVEL es MIA, la longitud de la variable del lenguaje principal *m* es igual al número de caracteres que preceden al primer terminador nulo de gráficos.
- No se soportan los tipos de datos numéricos sin signo.
- El tipo de datos `int` de C/C++ no está permitido, puesto que su representación interna depende de la máquina.

Conceptos relacionados:

- "Sección declare de SQL con variables del lenguaje principal para C y C++" en la página 178

FOR BIT DATA en C y C++

No se debe utilizar el tipo de serie estándar de C o C++, 460, para las columnas designadas como FOR BIT DATA. El gestor de bases de datos trunca este tipo de datos cuando se encuentra un carácter nulo. Utilice las estructuras VARCHAR (tipo de SQL 448) o CLOB (tipo de SQL 408).

Conceptos relacionados:

- "Sección declare de SQL con variables del lenguaje principal para C y C++" en la página 178

Información relacionada:

- "Tipos de datos SQL soportados en C y C++" en la página 180

Tipos de datos C y C++ para procedimientos, funciones y métodos

La tabla siguiente lista las correlaciones soportadas entre tipos de datos de SQL y tipos de datos de C para procedimientos, UDF y métodos.

Tabla 14. Tipos de datos de SQL correlacionados con declaraciones C/C++

Tipo de columna de SQL	Tipo de datos de C/C++	Descripción del tipo de columna de SQL
SMALLINT (500 ó 501)	short	Entero con signo de 16 bits
INTEGER (496 ó 497)	sqlint32	Entero con signo de 32 bits
BIGINT (492 ó 493)	sqlint64	entero con signo de 64 bits
REAL (480 ó 481)	float	Coma flotante de precisión simple
DOUBLE (480 ó 481)	double	Coma flotante de precisión doble
DECIMAL(*p*,*s*) (484 ó 485)	No soportado.	Para pasar un valor decimal, defina el parámetro como tipo de datos convertible desde DECIMAL (por ejemplo, CHAR o DOUBLE) y convierta explícitamente el argumento a este tipo.

Tabla 14. Tipos de datos de SQL correlacionados con declaraciones C/C++ (continuación)

Tipo de columna de SQL	Tipo de datos de C/C++	Descripción del tipo de columna de SQL
CHAR(*n*) (452 ó 453)	char[*n+1*] donde n es suficientemente grande para contener los datos 1<=*n*<=254	Serie de caracteres de longitud fija terminada en nulo
CHAR(*n*) FOR BIT DATA (452 ó 453)	char[*n+1*] donde n es suficientemente grande para contener los datos 1<=*n*<=254	Serie de caracteres de longitud fija
VARCHAR(*n*) (448 ó 449) (460 ó 461)	char[*n+1*] donde n es suficientemente grande para contener los datos 1<=*n*<=32 672	Serie de longitud variable terminada en nulo

VARCHAR(*n*) FOR BIT DATA
(448 ó 449)
	struct {
	sqluint16 length;
	char[*n*]
	}
	
	1<=*n*<=32 672
		Serie de caracteres de longitud variable no terminada en nulo

LONG VARCHAR
(456 ó 457)
	struct {
	sqluint16 length;
	char[*n*]
	}
	
	32 673<=*n*<=32 700
		Serie de caracteres de longitud variable no terminada en nulo

CLOB(*n*)
(408 ó 409)
	struct {
	sqluint32 length;
	char data[n];
	}
	
	1<=*n*<=2 147 483 647
		Serie de caracteres de longitud variable no terminada en nulo con indicador de longitud de serie de 4 bytes

BLOB(*n*)
(404 ó 405)
	struct {
	sqluint32 length;
	char data[n];
	}
	
	1<=*n*<=2 147 483 647
		Serie binaria de longitud variable no terminada en nulo con indicador de longitud de serie de 4 bytes

DATE (384 ó 385)	char[11]	Formato de caracteres terminado en nulo
TIME (388 ó 389)	char[9]	Formato de caracteres terminado en nulo
TIMESTAMP (392 ó 393)	char[27]	Formato de caracteres terminado en nulo
Nota: Los tipos de datos siguientes sólo están disponibles en los entornos DBCS o EUC si se compilan previamente con la opción WCHARTYPE NOCONVERT.		
GRAPHIC(*n*) (468 ó 469)	sqldbchar[*n+1*] donde n es suficientemente grande para contener los datos 1<=*n*<=127	Serie de caracteres de doble byte y longitud fija terminada en nulo
VARGRAPHIC(*n*) (400 ó 401)	sqldbchar[*n+1*] donde n es suficientemente grande para contener los datos 1<=*n*<=16 336	Serie de caracteres de doble byte y longitud variable no terminada en nulo

LONG VARGRAPHIC
(472 ó 473)
	struct {
	sqluint16 length;
	sqldbchar[*n*]
	}
	
	16 337<=*n*<=16 350
		Serie de caracteres de doble byte y longitud variable no terminada en nulo

Tabla 14. Tipos de datos de SQL correlacionados con declaraciones C/C++ (continuación)

Tipo de columna de SQL	Tipo de datos de C/C++	Descripción del tipo de columna de SQL

DBCLOB(*n*)
(412 ó 413)
	struct {
	sqluint32 length;
	sqldbchar data[n];
	}
	
	1<=*n*<=1 073 741 823
		Serie de caracteres de longitud variable no terminada en nulo con indicador de longitud de serie de 4 bytes

Variables SQLSTATE y SQLCODE en C y C++

Cuando se utiliza la opción de precompilación LANGLEVEL con un valor de SQL92E, se pueden incluir las dos declaraciones siguientes como variables del lenguaje principal:

```
EXEC SQL BEGIN DECLARE SECTION;
  char      SQLSTATE[6]
  sqlint32  SQLCODE;


⋮
  EXEC SQL END DECLARE SECTION;
```

Si no se especifica ninguna de ellas, se supone la declaración SQLCODE durante el paso de precompilación. Observe que, si se utiliza esta opción, no se debe especificar la sentencia INCLUDE SQLCA.

En una aplicación formada por varios archivos fuente, las variables SQLCODE y SQLSTATE se pueden definir en el primer archivo fuente, tal como en el ejemplo anterior. Los archivos fuente posteriores deben modificar las definiciones del modo siguiente:

```
extern sqlint32 SQLCODE;
extern char     SQLSTATE[6];
```

Conceptos relacionados:

- "Códigos de retorno" en la página 110
- "Información de error en los campos SQLCODE, SQLSTATE y SQLWARN" en la página 110

Capítulo 7. Acceso a bases de datos de varias hebras en aplicaciones C y C++

Objetivo del acceso a base de datos de varias hebras 187
Recomendaciones para utilizar varias hebras . . . 188
Consideraciones sobre página de códigos y código de país/región para aplicaciones UNIX de varias hebras 189
Resolución de problemas de aplicaciones de varias hebras 190
Problemas potenciales con varias hebras . . . 190
Cómo evitar puntos muertos para varios contextos 190

Objetivo del acceso a base de datos de varias hebras

Una característica de algunos sistemas operativos es la posibilidad de ejecutar varias hebras de ejecución en un solo proceso. Las diversas hebras permiten que una aplicación maneje sucesos asíncronos y facilitan la creación de aplicaciones dirigidas por sucesos sin recurrir a esquemas de tramas. La siguiente información explica cómo trabaja el gestor de bases de datos con varias hebras y se relacionan algunas directrices de diseño que conviene recordar.

Si no está familiarizado con los términos relacionados con el desarrollo de aplicaciones de varias hebras (como sección crítica y semáforo), consulte la documentación sobre programación correspondiente al sistema operativo.

Una aplicación DB2 puede ejecutar sentencias de SQL para varias hebras utilizando *contextos*. Un contexto es el entorno desde el que una aplicación ejecuta todas las sentencias de SQL y las llamadas a las API. Todas las conexiones, unidades de trabajo y otros recursos de base de datos están asociados a un contexto específico. Cada contexto está asociado a una o más hebras de una aplicación.

Para cada sentencia de SQL ejecutable en un contexto, la primera llamada a los servicios de tiempo de ejecución siempre intenta obtener un enganche. Si esta operación resulta satisfactoria, prosigue el proceso. Si no lo es (porque una sentencia de SQL de otra hebra del mismo contexto ya tiene el enganche), se bloquea la llamada en un semáforo de señalización hasta que entra en funciones dicho semáforo, en cuyo momento la llamada obtiene el enganche y prosigue el proceso. Se mantiene en enganche hasta que se ha completado el proceso, momento en que lo libera la última llamada a los servicios de tiempo de ejecución que se ha generado para esa sentencia de SQL en particular.

El resultado neto es que cada sentencia de SQL de un contexto se ejecuta como una unidad atómica, aunque puede que haya otras hebras que estén intentando ejecutar sentencias de SQL al mismo tiempo. Esta acción asegura que las distintas hebras que puedan actuar a la vez no alteran las estructuras internas de los datos. Las API también utilizan el enganche utilizado por los servicios de tiempo de ejecución; por consiguiente, las API tienen las mismas restricciones que las rutinas de los servicios de tiempo de ejecución dentro de cada contexto.

Para DB2® Versión 8, todas las aplicaciones de Versión 8 tienen varias hebras por omisión y pueden utilizar varios contextos. (El comportamiento de las aplicaciones anteriores a la Versión 8 no se modificará.) Si lo desea puede utilizar las API de DB2 siguientes para utilizar varios contextos. Específicamente, su aplicación puede crear un contexto para una hebra, conectarse o desconectarse de un contexto

independiente para cada hebra y pasar contextos entre las hebras. Si su aplicación no llama a *ninguna* de estas API, DB2 administrará automáticamente los contextos múltiples para sus aplicaciones:

- `sqleBeginCtx()`
- `sqleEndCtx()`
- `sqleAttachToCtx()`
- `sqleDetachFromCtx()`
- `sqleGetCurrentCtx()`
- `sqleInterruptCtx()`

En un proceso, se pueden intercambiar contextos entre hebras, pero no se pueden intercambiar entre procesos. Un uso de varios contextos consiste en proporcionar soporte de transacciones simultáneas.

Conceptos relacionados:

- "Transacciones simultáneas" en la página 678

Información relacionada:

- "sqleAttachToCtx - Attach to Context" en la publicación *Administrative API Reference*
- "sqleBeginCtx - Create and Attach to an Application Context" en la publicación *Administrative API Reference*
- "sqleDetachFromCtx - Detach From Context" en la publicación *Administrative API Reference*
- "sqleEndCtx - Detach and Destroy Application Context" en la publicación *Administrative API Reference*
- "sqleGetCurrentCtx - Get Current Context" en la publicación *Administrative API Reference*
- "sqleInterruptCtx - Interrupt Context" en la publicación *Administrative API Reference*

Ejemplos relacionados:

- "dbthrds.sqc -- How to use multiple context APIs on UNIX (C)"
- "dbthrds.sqC -- How to use multiple context APIs on UNIX (C++)"

Recomendaciones para utilizar varias hebras

Cuando acceda a una base de datos desde aplicaciones de varias hebras, siga estas directrices:

- Coloque en serie la alteración de estructuras de datos.

  Las aplicaciones se deben asegurar de que las estructuras de datos, definidas por el usuario y utilizadas por las sentencias de SQL y por las rutinas del gestor de bases de datos, no se vean alteradas por una hebra mientras se esté procesando una sentencia de SQL o una rutina del gestor de bases de datos en otra hebra. Por ejemplo, no permita que una hebra reasigne una SQLDA mientras la está utilizando una sentencia de SQL en otra hebra.

- Considere la posibilidad de utilizar estructuras de datos separadas.

  Puede resultar más fácil asignar a cada hebra sus propias estructuras de datos definidas por el usuario, a fin de evitar que se coloque en serie su uso. Esta directriz es especialmente cierta para la SQLCA, la cual utilizan no sólo todas las sentencias de SQL ejecutables, sino también todas las rutinas del gestor de bases de datos. Existen tres alternativas para evitar este problema con la SQLCA:

– Utilice EXEC SQL INCLUDE SQLCA, pero añada `struct sqlca sqlca` al comienzo de cualquier rutina que utilice cualquier hebra que no sea la primera.
– Coloque EXEC SQL INCLUDE SQLCA dentro de cada rutina que contenga SQL, en lugar de hacerlo en el ámbito global.
– Sustituya EXEC SQL INCLUDE SQLCA por `#include "sqlca.h"` y luego añada `"struct sqlca sqlca"` al comienzo de cualquier rutina que utilice SQL.

Nota: Se recomienda no utilizar el tamaño de pila por omisión, en su lugar aumente el tamaño de la pila para que como mínimo sea 256 000. DB2® requiere un tamaño de pila mínimo de 256 000 al llamar a una función de DB2. Por tanto, debe asegurarse de que asigna un tamaño de pila total que sea lo suficientemente grande para la aplicación y los requisitos mínimos para una llamada de función de DB2.

Consideraciones sobre página de códigos y código de país/región para aplicaciones UNIX de varias hebras

En AIX®, el Entorno operativo Solaris, HP-UX y Silicon Graphics IRIX, se han efectuado cambios en las funciones que se utilizan para consultas de la página de códigos y del código de país/región que se deben utilizar para conectar con una base de datos. Ahora estas funciones tienen ejecución protegida de hebras, pero pueden crear alguna contención de bloqueo (y la consiguiente degradación del rendimiento) en una aplicación multihebra que utilice muchas conexiones de base de datos simultáneas.

Puede utilizar la variable de entorno DB2_FORCE_NLS_CACHE para eliminar la posibilidad de contención de bloqueos en las aplicaciones de varias hebras. Si DB2_FORCE_NLS_CACHE tiene el valor TRUE, la primera vez que una hebra accede a la información de página de códigos y código de país/región, ésta se guarda. A partir de este punto, cualquier otra hebra que solicite esta información utilizará la que se encuentra en la antememoria. Guardando esta información, se suprime la contención de bloqueos y, en determinadas situaciones, se obtiene una mejora en el rendimiento.

No debe asignar a DB2_FORCE_NLS_CACHE el valor TRUE si la aplicación cambia los valores de entorno nacional entre conexiones. Si se produce esta situación, se devolverá la información del entorno nacional original aunque se hayan cambiado estos valores. En general, las aplicaciones de varias hebras no cambiarán los valores de entorno nacional, lo cual asegura que la aplicación sigue estando a salvo de hebras.

Conceptos relacionados:

- "DB2 registry and environment variables" en la publicación *Administration Guide: Performance*

Resolución de problemas de aplicaciones de varias hebras

Las secciones siguientes describen problemas que se pueden producir con una aplicación de varias hebras y cómo evitarlos.

Problemas potenciales con varias hebras

Una aplicación que utiliza varias hebras es, comprensiblemente, más compleja que una aplicación de una sola hebra. Esta complejidad adicional puede conducir potencialmente a algunos problemas inesperados. Cuando escriba una aplicación de varias hebras, tenga precaución con lo siguiente:

- Dependencias de base de datos entre dos o más contextos.

  Cada contexto de una aplicación tiene sus propios recursos de base de datos, incluidos los bloqueos sobre objetos de base de datos. Esta característica posibilita que dos contextos, si están accediendo al mismo objeto de base de datos, lleguen a un punto muerto. El gestor de bases de datos detectará el punto muerto. Uno de los contextos recibirá SQLCODE -911 y se retrotraerá la unidad de trabajo del mismo.
- Dependencias de aplicaciones entre dos o más contextos.

  Tenga cuidado con las técnicas de programación que establecen dependencias entre contextos. Los enganches, los semáforos y las secciones críticas son ejemplos de técnicas de programación que pueden establecer dichas dependencias. Si una aplicación tiene dos contextos que tienen dependencias, tanto de aplicación como de base de datos, entre los contextos, es posible que la aplicación llegue a un punto muerto. Si algunas de las dependencias están fuera del gestor de bases de datos, no se detecta el punto muerto, por lo que la aplicación se suspende o se cuelga.

Conceptos relacionados:

- "Cómo evitar puntos muertos para varios contextos" en la página 190

Cómo evitar puntos muertos para varios contextos

Puesto que el gestor de bases de datos no detecta puntos muertos entre hebras, diseñe y codifique la aplicación de modo que impida (o al menos evite) puntos muertos.

Como ejemplo de un punto muerto que el gestor de bases de datos no detectaría piense en una aplicación que tiene dos contextos y ambos acceden a una estructura de datos común. Para evitar problemas en que ambos contextos cambien simultáneamente la estructura de datos, la estructura de datos se protege mediante un semáforo. Los contextos tienen el aspecto siguiente:

```
contexto 1
SELECT * FROM TAB1 FOR UPDATE....
UPDATE TAB1 SET....
get semaphore
access data structure
release semaphore
COMMIT

contexto 2
get semaphore
access data structure
SELECT * FROM TAB1...
release semaphore
COMMIT
```

Suponga que el primer contexto ejecuta satisfactoriamente las sentencias SELECT y UPDATE, mientras que el segundo contexto obtiene el semáforo y accede a la estructura de datos. Ahora, el primer contexto intenta obtener el semáforo, pero no puede porque el segundo contexto lo está reteniendo. A continuación, el segundo contexto intenta leer una fila de la tabla TAB1, pero se detiene en un bloqueo de la base de datos mantenido por el primer contexto. La aplicación se encuentra ahora en un estado en que el contexto 1 no puede terminar antes de que lo haga el contexto 2, y el contexto 2 está esperando a que el contexto 1 termine. La aplicación está en un punto muerto pero, puesto que el gestor de bases de datos no sabe nada de la dependencia del semáforo, no se retrotraerá ninguno de los contextos. La dependencia no resuelta deja la aplicación suspendida.

Puede evitar el punto muerto que se produciría en el ejemplo anterior de varias formas.

- Libere todos los bloqueos mantenidos antes de obtener el semáforo.

  Cambie el código del contexto 1 de forma que realice una confirmación antes de obtener el semáforo.
- No codifique sentencias de SQL en una sección protegida por semáforos.

  Cambie el código del contexto 2 de forma que libere el semáforo antes de ejecutar SELECT.
- Codifique todas las sentencias de SQL dentro de semáforos.

  Cambie el código del contexto 1 de forma que obtenga el semáforo antes de ejecutar la sentencia SELECT. Aunque esta técnica funcionará, no es muy recomendable, puesto que los semáforos colocarán en serie el acceso al gestor de bases de datos, lo cual invalidará potencialmente las ventajas de utilizar varias hebras.
- Establezca el parámetro de configuración de la base de datos *locktimeout* en un valor distinto de -1.

  Aunque un valor distinto de -1 no impedirá el punto muerto, permitirá que se reanude la ejecución. Eventualmente, se retrotraerá el contexto 2, puesto que no puede obtener el bloqueo solicitado. Cuando maneje el error de retrotracción, el contexto 2 debe liberar el semáforo. Una vez que se haya liberado el semáforo, el contexto 1 podrá continuar y el contexto 2 será libre de reintentar su trabajo.

Las técnicas para evitar puntos muertos se describen en términos del ejemplo, pero las puede aplicar a todas las aplicaciones de varias hebras. En general, trate el gestor de bases de datos tal como trataría cualquier recurso protegido y no experimentará problemas con las aplicaciones de varias hebras.

Conceptos relacionados:

- “Problemas potenciales con varias hebras” en la página 190

Capítulo 8. Programación en COBOL

Consideraciones sobre la programación en COBOL 193
Restricciones de lenguaje en COBOL 193
Acceso a bases de datos de varias hebras en COBOL 193
Archivos de entrada y salida para COBOL . . . 194
Archivos include para COBOL 194
Sentencias de SQL incorporado en COBOL . . . 196
Variables del lenguaje principal en COBOL . . . 198
Variables del lenguaje principal en COBOL . . 198
Nombres de variables del lenguaje principal en COBOL 199
Declaraciones de variables del lenguaje principal en COBOL 200
Sintaxis de las variables numéricas del lenguaje principal en COBOL 200
Sintaxis de las variables del lenguaje principal de tipo carácter de longitud fija en COBOL . . 201
Sintaxis de las variables gráficas del lenguaje principal de longitud fija en COBOL 203
Variables de indicador en COBOL 204
Sintaxis de las variables del lenguaje principal LOB en COBOL 204
Sintaxis de las variables del lenguaje principal de localizador de LOB en COBOL 205
Sintaxis de las variables del lenguaje principal de referencia de archivos en COBOL 206
Soporte de estructura del lenguaje principal en COBOL 206
Tablas de indicadores en COBOL. 208
REDEFINES en elementos de datos de grupos COBOL 209
Sección declare de SQL con variables del lenguaje principal para COBOL 209
Consideraciones sobre tipos de datos para COBOL 210
Tipos de datos de SQL soportados en COBOL 210
Tipos de datos BINARY/COMP-4 de COBOL 213
FOR BIT DATA en COBOL. 213
Variables SQLSTATE y SQLCODE en COBOL . . 213
Consideraciones sobre EUC en japonés o chino tradicional y UCS-2 para COBOL. 213
COBOL orientado a objetos. 214

Consideraciones sobre la programación en COBOL

En las secciones siguientes se explican las consideraciones especiales sobre la programación en el lenguaje principal. Se incluye información sobre las restricciones de lenguaje, los archivos include específicos del lenguaje principal, la incorporación de sentencias de SQL, las variables del lenguaje principal y los tipos de datos soportados para las variables del lenguaje principal. Para obtener información sobre cómo incorporar sentencias de SQL, las restricciones del lenguaje y los tipos de datos soportados para las variables del lenguaje principal, consulte la documentación de Micro Focus COBOL.

Información relacionada:

- "Ejemplos de COBOL" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Restricciones de lenguaje en COBOL

Todos los punteros de API tienen una longitud de 4 bytes. Todas las variables enteras utilizadas como parámetros de valor en las llamadas a las API se deben declarar con una cláusula USAGE COMP-5.

Acceso a bases de datos de varias hebras en COBOL

COBOL no da soporte al acceso a bases de datos de varias hebras.

Archivos de entrada y salida para COBOL

Por omisión, el archivo de entrada tiene la extensión `.sqb`, pero si se utiliza la opción de precompilación TARGET (TARGET ANSI_COBOL, TARGET IBMCOB, TARGET MFCOB o TARGET MFCOB16), el archivo de entrada puede tener la extensión que elija el usuario.

Por omisión, el archivo de salida tiene la extensión `.cbl`, pero se puede utilizar la opción de precompilación OUTPUT para especificar un nombre y una vía de acceso nuevos para el archivo fuente de salida modificado.

Archivos include para COBOL

Los archivos include específicos del lenguaje principal para COBOL tienen la extensión de archivo `.cbl`. Si se utiliza la característica ″Soporte para tipo de datos de sistema principal System/390″ del compilador COBOL de IBM, los archivos include de DB2 para las aplicaciones se encuentran en el directorio siguiente:

```
$HOME/sqllib/include/cobol_i
```

Si crea los programas de ejemplo de DB2 con los archivos de script suministrados, deben cambiar la vía de acceso de archivos include especificada en los archivos de script por el directorio `cobol_i`, y no por el directorio `cobol_a`.

Si **no** utiliza la característica ″Soporte para tipo de datos de sistema principal System/390″ del compilador COBOL de IBM, o si utiliza una versión anterior de este compilador, los archivos include de DB2 para las aplicaciones se encuentran en el directorio siguiente:

```
$HOME/sqllib/include/cobol_a
```

A continuación se describen los archivos include destinados a su uso en las aplicaciones del usuario.

SQL (sql.cbl) Este archivo incluye prototipos específicos del lenguaje para el vinculador, el precompilador y las API de recuperación de mensajes de error. También define constantes del sistema.

SQLAPREP (sqlaprep.cbl)
Este archivo contiene definiciones necesarias para escribir su propio precompilador.

SQLCA (sqlca.cbl)
Este archivo define la estructura del Área de comunicaciones de SQL (SQLCA). El SQLCA contiene variables que utiliza el gestor de bases de datos para proporcionar a una aplicación información de error sobre la ejecución de sentencias de SQL y llamadas a API.

SQLCA_92 (sqlca_92.cbl)
Este archivo contiene una versión que se ajusta a FIPS SQL92 Entry Level de la estructura Área de comunicaciones de SQL (SQL Communications Area (SQLCA)). Debe incluir este archivo en lugar del archivo `sqlca.cbl` cuando escriba aplicaciones DB2 que se ajusten al estándar FIPS SQL92 Entry Level. El precompilador de DB2 incluye automáticamente el archivo `sqlca_92.cbl` cuando la opción LANGLEVEL del precompilador se establece en SQL92E.

SQLCODES (sqlcodes.cbl)
Este archivo define constantes para el campo SQLCODE de la estructura SQLCA.

SQLDA (sqlda.cbl)
Este archivo define la estructura del Área de descriptores de SQL (SQLDA). La SQLDA se utiliza para pasar datos entre una aplicación y el gestor de bases de datos.

SQLEAU (sqleau.cbl)
Este archivo contiene definiciones de constantes y de estructuras necesarias para las API de auditoría de seguridad de DB2. Si utiliza estas API, tiene que incluir este archivo en el programa. Este archivo también contiene definiciones de constantes y de valores de palabras clave para los campos del registro de seguimiento de auditoría. Programas externos o de extracción de seguimiento de auditoría de proveedores pueden utilizar estas definiciones.

SQLENV (sqlenv.cbl)
Este archivo define llamadas específicas del lenguaje para las API del entorno de bases de datos y las estructuras, constantes y códigos de retorno correspondientes a dichas interfaces.

SQLETSD (sqletsd.cbl)
Este archivo define la estructura Descriptor de espacio de tablas (Table Space Descriptor), SQLETSDESC, que se pasa a la API para Crear base de datos, sqlgcrea.

SQLE819A (sqle819a.cbl)
Si la página de códigos de la base de datos es 819 (ISO Latin-1), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 500 (EBCDIC internacional) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLE819B (sqle819b.cbl)
Si la página de códigos de la base de datos es 819 (ISO Latin-1), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 037 (EBCDIC inglés de EE.UU.) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLE850A (sqle850a.cbl)
Si la página de códigos de la base de datos es 850 (ASCII Latin-1), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 500 (EBCDIC internacional) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLE850B (sqle850b.cbl)
Si la página de códigos de la base de datos es 850 (ASCII Latin-1), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 037 (EBCDIC inglés de EE.UU.) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLE932A (sqle932a.cbl)
Si la página de códigos de la base de datos es 932 (ASCII japonés), esta secuencia clasifica series de caracteres que no son FOR BIT

DATA según la clasificación binaria CCSID 5035 (EBCDIC japonés) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLE932B (sqle932b.cbl)
Si la página de códigos de la base de datos es 932 (ASCII japonés), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 5026 (EBCDIC japonés) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQL1252A (sql1252a.cbl)
Si la página de códigos de la base de datos es 1252 (Windows Latin-1), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 500 (EBCDIC internacional) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQL1252B (sql1252b.cbl)
Si la página de códigos de la base de datos es 1252 (Windows Latin-1), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 037 (EBCDIC inglés de EE.UU.) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLMON (sqlmon.cbl)
Este archivo define llamadas específicas del lenguaje para las API del supervisor del sistema de bases de datos y las estructuras, constantes y códigos de retorno correspondientes a dichas interfaces.

SQLMONCT (sqlmonct.cbl)
Este archivo contiene definiciones de constantes y definiciones de estructuras de datos locales necesarias para llamar a las API del Supervisor del sistema de bases de datos.

SQLSTATE (sqlstate.cbl)
Este archivo define constantes correspondientes al campo SQLSTATE de la estructura SQLCA.

SQLUTBCQ (sqlutbcq.cbl)
Este archivo define la estructura de datos (Consulta de contenedor de espacio de tablas (Table Space Container Query), SQLB-TBSCONTQRY-DATA, que se utiliza con las API de consulta del contenedor del espacio de tablas, sqlgstsc, sqlgftcq y sqlgtcq.

SQLUTBSQ (sqlutbsq.cbl)
Este archivo define la estructura de datos Consulta de espacio de tablas (Table Space Query), SQLB-TBSTQRY-DATA, que se utiliza con las API de consulta del espacio de tablas, sqlgstsq, sqlgftsq y sqlgtsq.

SQLUTIL (sqlutil.cbl)
Este archivo define las llamadas específicas del lenguaje correspondientes a las API de programas de utilidad y las estructuras, constantes y códigos necesarios para dichas interfaces.

Sentencias de SQL incorporado en COBOL

Las sentencias de SQL incorporado constan de los tres elementos siguientes:

Elemento	Sintaxis correcta en COBOL
Par de palabras clave	EXEC SQL
Serie de la sentencia	Cualquier sentencia de SQL válida
Terminador de la sentencia	END-EXEC.

Por ejemplo:

```
EXEC SQL SELECT col INTO :hostvar FROM table END-EXEC.
```

Se aplican las normas siguientes a las sentencias de SQL incorporado:

- Las sentencias de SQL ejecutables se deben colocar en PROCEDURE DIVISION. Las sentencias de SQL pueden ir precedidas de un nombre de párrafo, al igual que una sentencia de COBOL.
- Las sentencias de SQL puede empezar en el Área A (columnas 8 a 11) o en el Área B (columnas 12 a 72).
- Inicie cada sentencia de SQL con EXEC SQL y termínela con END-EXEC. El precompilador SQL incluye cada una de las sentencias de SQL como comentarios en el archivo fuente modificado.
- Debe utilizar el terminador de sentencias de SQL. De no utilizarlo, el precompilador seguirá hasta el siguiente terminador de la aplicación. Esto puede producir errores indeterminados.
- Están permitidos los comentarios de SQL en cualquier línea que forme parte de una sentencia de SQL incorporado. Estos comentarios no están permitidos en las sentencias que se ejecutan de forma dinámica. El formato de un comentario de SQL consiste en un guión doble (`--`), seguido de una serie compuesta por cero o más caracteres y terminada por un fin de línea. No coloque comentarios de SQL detrás del terminador de la sentencia de SQL, puesto que ocasionarían errores de compilación debido a que parecería que forman parte del lenguaje COBOL.
- Los comentarios COBOL están permitidos *casi* en cualquier lugar de una sentencia de SQL incorporado. Las excepciones son:
  - No se permiten comentarios entre EXEC y SQL.
  - No se permiten comentarios en las sentencias que se ejecutan dinámicamente.
- Las sentencias de SQL siguen las mismas normas de continuación de línea que el lenguaje COBOL. No obstante, no divida el par de palabras clave EXEC SQL en distintas líneas.
- No utilice la sentencia COPY de COBOL para incluir archivos que contengan sentencias de SQL. Las sentencias de SQL se precompilan antes de que se compile el módulo. El precompilador pasará por alto la sentencia COPY de COBOL. En su lugar, utilice la sentencia INCLUDE de SQL para incluir dichos archivos.

  Para localizar el archivo INCLUDE, el precompilador COBOL de DB2® busca en primer lugar en el directorio actual y, a continuación, en los directorios especificados por la variable de entorno DB2INCLUDE. Considere los ejemplos siguientes:

  - `EXEC SQL INCLUDE payroll END-EXEC.`

    Si el archivo especificado en la sentencia INCLUDE no está encerrado entre comillas, como en el ejemplo anterior, el precompilador C busca `payroll.sqb`, luego `payroll.cpy` y luego `payroll.cbl` en cada uno de los directorios que mira.
  - `EXEC SQL INCLUDE 'pay/payroll.cbl' END-EXEC.`

    Si el nombre de archivo está encerrado entre comillas, como en el caso anterior, no se añade ninguna extensión al nombre.

Si el nombre del archivo entrecomillado no contiene una vía de acceso absoluta, se utiliza el contenido de DB2INCLUDE para buscar el archivo, añadiéndole como prefijo la vía de acceso especificada en el nombre del archivo INCLUDE. Por ejemplo, con DB2 para AIX, si DB2INCLUDE se establece en '/disk2:myfiles/cobol', el precompilador busca './pay/payroll.cbl', luego '/disk2/pay/payroll.cbl' y finalmente './myfiles/cobol/pay/payroll.cbl'. En los mensajes del precompilador se muestra la vía de acceso en la que se encuentra realmente el archivo. En las plataformas Windows®, sustituya las barras inclinadas invertidas (\) del ejemplo anterior por barras inclinadas.

Nota: El procesador de línea de mandatos de DB2 coloca en antememoria el valor de DB2INCLUDE. Para cambiar el valor de DB2INCLUDE después de haber emitido mandatos del CLP, entre el mandato TERMINATE, luego vuelva a conectar con la base de datos y realice una precompilación del modo habitual.

- Para continuar una constante de serie en la línea siguiente, en la columna 7 de la línea de continuación debe aparecer un '-' y en la columna 12 o posteriores debe aparecer un delimitador de serie.
- Los operadores aritméticos de SQL se deben delimitar mediante espacios en blanco.
- Pueden aparecer comentarios de COBOL de línea completa en cualquier lugar del programa, incluso dentro de sentencias de SQL.
- Cuando haga referencia a variables del lenguaje principal en una sentencia de SQL, utilícelas exactamente tal como se han declarado.
- La sustitución de caracteres de espacio en blanco, como caracteres de fin de línea y de tabulación, se produce del siguiente modo:
  - Cuando aparecen fuera de las comillas (pero dentro de sentencias de SQL), los caracteres de fin de línea y tabuladores se sustituyen por un solo espacio.
  - Si aparecen dentro de las comillas, los caracteres de fin de línea desaparecen, siempre que se continúe correctamente la serie para un programa COBOL. Los tabuladores no se modifican.

  Observe que los caracteres reales utilizados como fin de línea y tabulador varían en función de la plataforma. Por ejemplo, las plataformas basadas en Windows utilizan el retorno de carro/salto de línea como fin de línea, mientras que los sistemas basados en UNIX® sólo utilizan simplemente un salto de línea.

Información relacionada:

- Apéndice A, "Sentencias de SQL soportadas", en la página 733

Variables del lenguaje principal en COBOL

Las secciones siguientes describen cómo declarar y utilizar variables del lenguaje principal en programas COBOL.

Variables del lenguaje principal en COBOL

Las variables del lenguaje principal son variables del lenguaje COBOL a las que se hace referencia en las sentencias de SQL. Permiten que una aplicación pase datos de entrada al gestor de bases de datos y reciba datos de salida de éste. Una vez que se ha precompilado la aplicación, el compilador utiliza las variables del lenguaje principal como cualquier otra variable de COBOL.

Conceptos relacionados:

- "Nombres de variables del lenguaje principal en COBOL" en la página 199
- "Declaraciones de variables del lenguaje principal en COBOL" en la página 200

Información relacionada:

- "Sintaxis de las variables numéricas del lenguaje principal en COBOL" en la página 200
- "Sintaxis de las variables del lenguaje principal de tipo carácter de longitud fija en COBOL" en la página 201
- "Sintaxis de las variables gráficas del lenguaje principal de longitud fija en COBOL" en la página 203
- "Sintaxis de las variables del lenguaje principal LOB en COBOL" en la página 204
- "Sintaxis de las variables del lenguaje principal de localizador de LOB en COBOL" en la página 205
- "Sintaxis de las variables del lenguaje principal de referencia de archivos en COBOL" en la página 206

Nombres de variables del lenguaje principal en COBOL

El precompilador SQL identifica las variables del lenguaje principal por el nombre declarado para éstas. Se aplican las normas siguientes:

- Especifique nombres de variables con una longitud máxima de 255 caracteres.
- Empiece los nombres de variables con prefijos que no sean `SQL`, `sql`, `DB2` ni `db2`, que están reservados para uso del sistema.
- Los elementos FILLER que utilizan las sintaxis de declaración descritas a continuación están permitidos en las declaraciones de variables del lenguaje principal de grupos y el precompilador los pasará por alto. Sin embargo, si se utiliza FILLER más de una vez dentro de una sección DECLARE de SQL, el precompilador fallará. No se pueden incluir elementos FILLER en declaraciones VARCHAR, LONG VARCHAR, VARGRAPHIC ni LONG VARGRAPHIC.
- Puede utilizar guiones en los nombres de variables del lenguaje principal.

  SQL interpreta un guión encerrado entre espacios como un operador de resta. Utilice guiones sin espacios en los nombres de variables del lenguaje principal.
- La cláusula REDEFINES está permitida en las declaraciones de variables del lenguaje principal.
- Las declaraciones de nivel 88 están permitidas en la sección de declaración de variables del lenguaje principal, pero se pasan por alto.

Conceptos relacionados:

- "Declaraciones de variables del lenguaje principal en COBOL" en la página 200

Información relacionada:

- "Sintaxis de las variables numéricas del lenguaje principal en COBOL" en la página 200
- "Sintaxis de las variables del lenguaje principal de tipo carácter de longitud fija en COBOL" en la página 201
- "Sintaxis de las variables gráficas del lenguaje principal de longitud fija en COBOL" en la página 203
- "Sintaxis de las variables del lenguaje principal LOB en COBOL" en la página 204

- "Sintaxis de las variables del lenguaje principal de localizador de LOB en COBOL" en la página 205
- "Sintaxis de las variables del lenguaje principal de referencia de archivos en COBOL" en la página 206

Declaraciones de variables del lenguaje principal en COBOL

Se debe utilizar una sección de declaración de SQL para identificar las declaraciones de variables del lenguaje principal. Esta sección alerta al precompilador acerca de las variables del lenguaje principal a las que se puede hacer referencia en sentencias de SQL posteriores.

El precompilador COBOL sólo reconoce un subconjunto de las declaraciones válidas de COBOL.

Tareas relacionadas:

- "Declaración de variables del lenguaje principal de tipo estructurado" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*

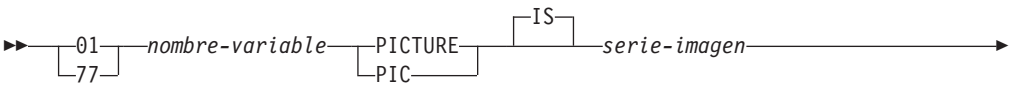
Información relacionada:

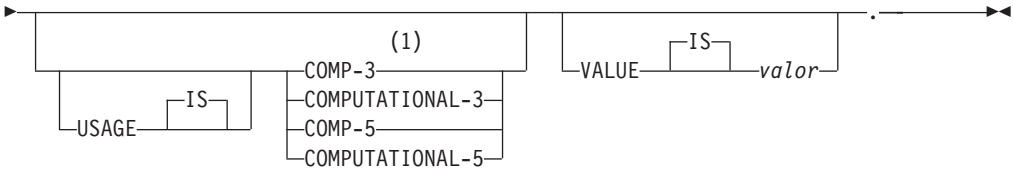
- "Sintaxis de las variables numéricas del lenguaje principal en COBOL" en la página 200
- "Sintaxis de las variables del lenguaje principal de tipo carácter de longitud fija en COBOL" en la página 201
- "Sintaxis de las variables gráficas del lenguaje principal de longitud fija en COBOL" en la página 203
- "Sintaxis de las variables del lenguaje principal LOB en COBOL" en la página 204
- "Sintaxis de las variables del lenguaje principal de localizador de LOB en COBOL" en la página 205
- "Sintaxis de las variables del lenguaje principal de referencia de archivos en COBOL" en la página 206

Sintaxis de las variables numéricas del lenguaje principal en COBOL

A continuación se muestra la sintaxis de las variables numéricas del lenguaje principal.

Sintaxis de las variables numéricas del lenguaje principal en COBOL

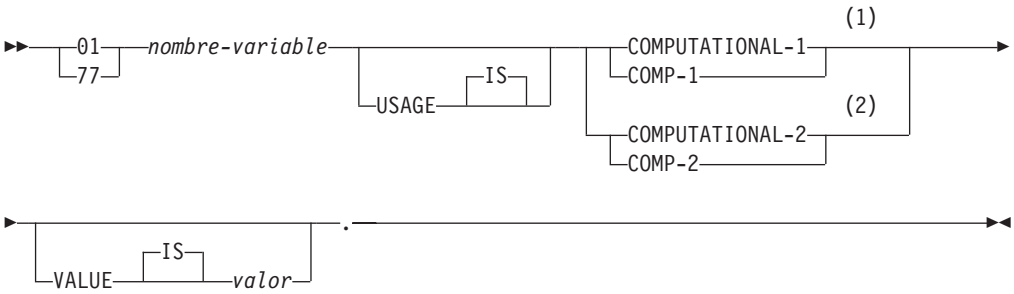




Notas:

1 Una alternativa de COMP-3 es PACKED-DECIMAL.

Coma flotante



Notas:

1 REAL (SQLTYPE 480), Longitud 4

2 DOUBLE (SQLTYPE 480), Longitud 8

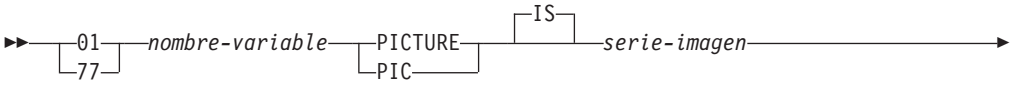
Consideraciones sobre las variables numéricas del lenguaje principal:

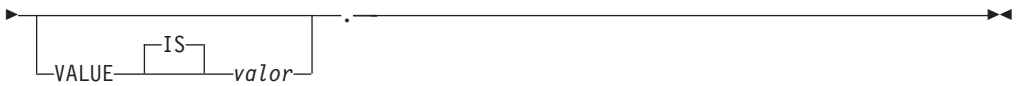
1. *Serie-imagen* debe tener uno de los formatos siguientes:
   - S9(m)V9(n)
   - S9(m)V
   - S9(m)
2. Los nueves se pueden expandir (por ejemplo, ″S999″ en lugar de S9(3)″)
3. *m* y *n* deben ser enteros positivos.

Sintaxis de las variables del lenguaje principal de tipo carácter de longitud fija en COBOL

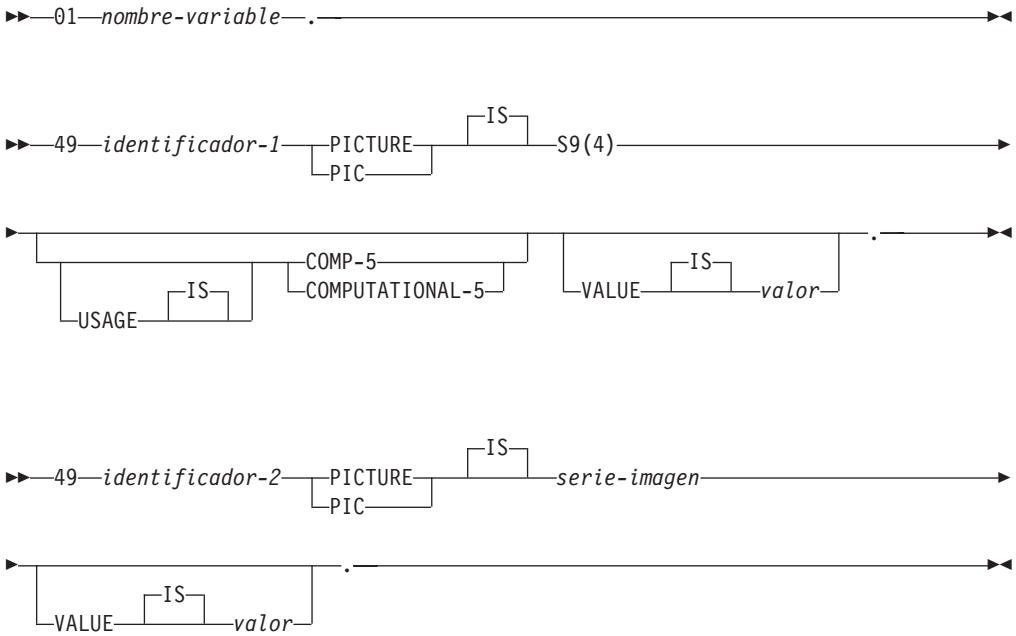
A continuación se muestra la sintaxis de las variables de tipo carácter del lenguaje principal.

Sintaxis de las variables del lenguaje principal de tipo carácter en COBOL: Longitud fija





Longitud variable



Consideraciones sobre las variables del lenguaje principal de tipo carácter:

1. *Serie-imagen* debe tener el formato *X(m)*. Como alternativa, se pueden expandir las X (por ejemplo, ″XXX″ en lugar de ″X(3)″).
2. *m* va de 1 a 254 para las series de longitud fija.
3. *m* va de 1 a 32 700 para las series de longitud variable.
4. Si *m* es mayor que 32 672, la variable del lenguaje principal se tratará como una serie LONG VARCHAR y su uso puede estar restringido.
5. Utilice X y 9 como caracteres de imagen en cualquier cláusula PICTURE. No están permitidos otros caracteres.
6. Las series de longitud variable constan de un elemento de longitud y un elemento de valor. Puede utilizar nombres aceptables de COBOL para el elemento de longitud y para el elemento de serie. No obstante, haga referencia a las series de longitud variable por el nombres colectivo en las sentencias de SQL.
7. En una sentencia CONNECT, como por ejemplo la que se muestra a continuación, a las variables del lenguaje principal de serie de caracteres en COBOL `dbname` y `userid` se les eliminarán los blancos de cola antes del proceso:

   ```
   EXEC SQL CONNECT TO :dbname USER :userid USING :p-word
   END-EXEC.
   ```

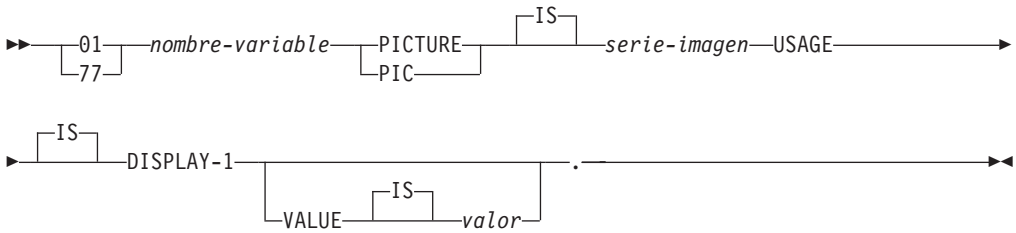
   Sin embargo, y puesto que los espacios en blanco pueden ser significativos en las contraseñas, la variable del lenguaje principal `p-word` se debe declarar como un elemento de datos VARCHAR, de forma que la aplicación pueda indicar explícitamente la longitud significativa de la contraseña para la sentencia CONNECT, del modo siguiente:

```
  EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 dbname PIC X(8).
01 userid PIC X(8).
01 p-word.
    49 L PIC S9(4) COMP-5.
    49 D PIC X(18).
EXEC SQL END DECLARE SECTION END-EXEC.
PROCEDURE DIVISION.
    MOVE "sample" TO dbname.
    MOVE "userid" TO userid.
    MOVE "password" TO D OF p-word.
    MOVE 8          TO L of p-word.
EXEC SQL CONNECT TO :dbname USER :userid USING :p-word
END-EXEC.
```

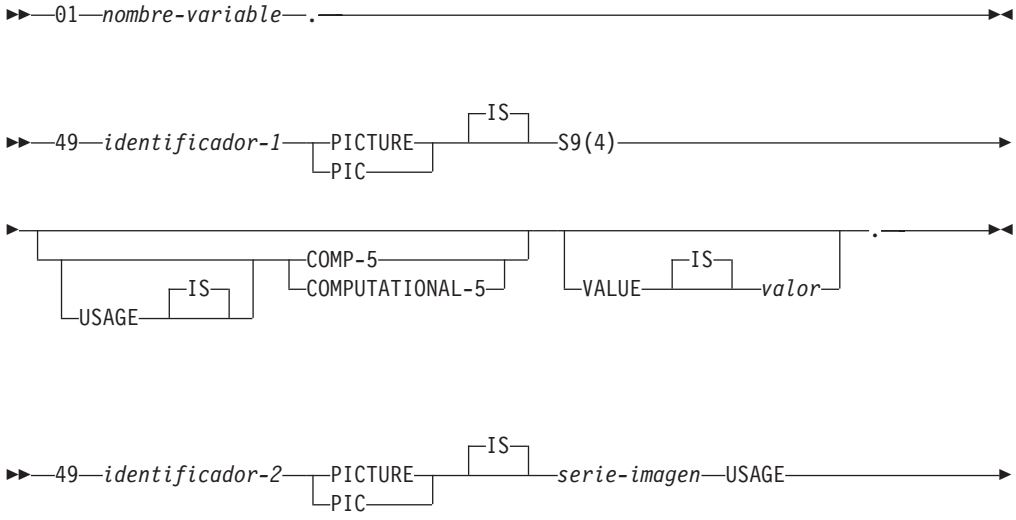
Sintaxis de las variables gráficas del lenguaje principal de longitud fija en COBOL

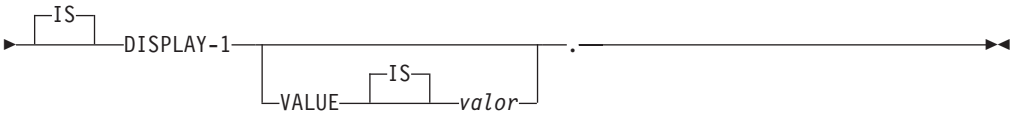
A continuación se muestra la sintaxis de las variables gráficas del lenguaje principal.

Sintaxis de las variables gráficas del lenguaje principal en COBOL: Longitud fija



Longitud variable





Consideraciones sobre las variables gráficas del lenguaje principal:

1. *Serie-imagen* debe tener el formato *G(m)*. Como alternativa, se pueden expandir las G (por ejemplo, ″GGG″ en lugar de ″G(3)″).
2. *m* va de 1 a 127 para las series de longitud fija.
3. *m* va de 1 a 16 350 para las series de longitud variable.
4. Si *m* es mayor que 16 336, la variable del lenguaje principal se tratará como una serie LONG VARGRAPHIC y su uso puede estar restringido.

Variables de indicador en COBOL

Las variables de indicador se deben declarar con tipo de datos `PIC S9(4) COMP-5`.

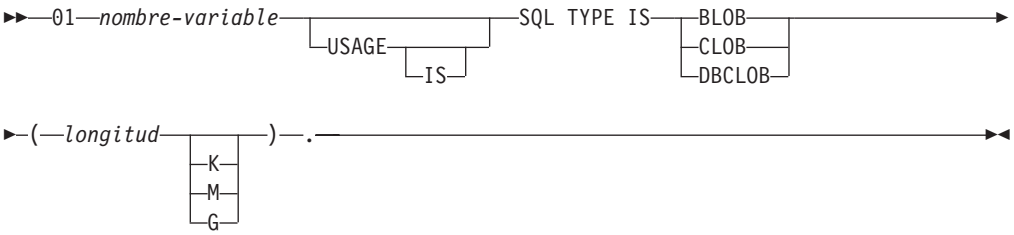
Conceptos relacionados:

- "Tablas de indicadores en COBOL" en la página 208

Sintaxis de las variables del lenguaje principal LOB en COBOL

A continuación se muestra la sintaxis para declarar variables del lenguaje principal de objeto grande (LOB) en COBOL.

Sintaxis de las variables del lenguaje principal LOB en COBOL



Consideraciones sobre las variables del lenguaje principal LOB:

1. Para BLOB y CLOB 1 <= longitud-lob <= 2 147 483 647.
2. Para DBCLOB 1 <= longitud-lob <= 1 073 741 823.
3. SQL TYPE IS, BLOB, CLOB, DBCLOB, K, M, G se pueden especificar en mayúsculas, en minúsculas o en una mezcla de ambas.
4. No se permite la inicialización dentro de la declaración LOB.
5. El nombre de la variable del lenguaje principal es el prefijo de LENGTH y DATA en el código generado por el precompilador.

Ejemplo de BLOB:

La declaración:

```
01 MY-BLOB USAGE IS SQL TYPE IS BLOB(2M).
```

Tiene como resultado la generación de la estructura siguiente:

```
01 MY-BLOB.
49 MY-BLOB-LENGTH PIC S9(9) COMP-5.
49 MY-BLOB-DATA PIC X(2097152).
```

Ejemplo de CLOB:

La declaración:

```
 01 MY-CLOB USAGE IS SQL TYPE IS CLOB(125M).
```

Tiene como resultado la generación de la estructura siguiente:

```
   01 MY-CLOB.
49 MY-CLOB-LENGTH PIC S9(9) COMP-5.
49 MY-CLOB-DATA PIC X(131072000).
```

Ejemplo de DBCLOB:

La declaración:

```
    01 MY-DBCLOB USAGE IS SQL TYPE IS DBCLOB(30000).
```

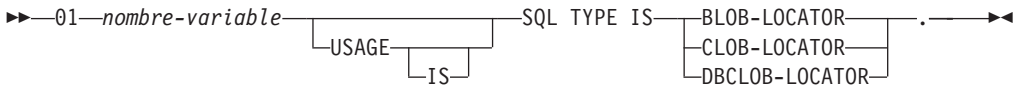
Tiene como resultado la generación de la estructura siguiente:

```
   01 MY-DBCLOB.
49 MY-DBCLOB-LENGTH PIC S9(9) COMP-5.
49 MY-DBCLOB-DATA PIC G(30000) DISPLAY-1.
```

Sintaxis de las variables del lenguaje principal de localizador de LOB en COBOL

A continuación se muestra la sintaxis para declarar variables del lenguaje principal de localizador de objeto grande (LOB) en COBOL.

Sintaxis de las variables del lenguaje principal



Consideraciones sobre las variables del lenguaje principal de localizador de LOB:

1. SQL TYPE IS, BLOB-LOCATOR, CLOB-LOCATOR, DBCLOB-LOCATOR se pueden especificar en mayúsculas, en minúsculas o en una mezcla de ambas.
2. No se permite la inicialización de localizadores.

Ejemplo de localizador de BLOB (existen otros tipos de localizadores de LOB parecidos):

La declaración:

```
01 MY-LOCATOR USAGE SQL TYPE IS BLOB-LOCATOR.
```

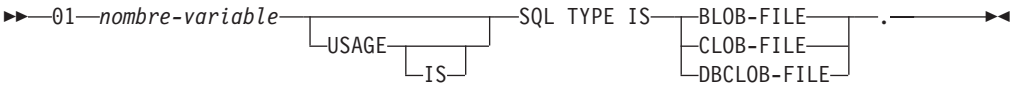
Tiene como resultado la generación de la declaración siguiente:

```
01 MY-LOCATOR PIC S9(9) COMP-5.
```

Sintaxis de las variables del lenguaje principal de referencia de archivos en COBOL

A continuación se muestra la sintaxis para declarar variables del lenguaje principal de referencia de archivos en COBOL.

Sintaxis de las variables del lenguaje principal de referencia de archivos en COBOL



- SQL TYPE IS, BLOB-FILE, CLOB-FILE, DBCLOB-FILE se pueden especificar en mayúsculas, en minúsculas o en una mezcla de ambas.

Ejemplo de referencia de archivos BLOB (existen otros tipos de LOB parecidos):

La declaración:

```
01 MY-FILE USAGE IS SQL TYPE IS BLOB-FILE.
```

Tiene como resultado la generación de la declaración siguiente:

```
01 MY-FILE.
   49 MY-FILE-NAME-LENGTH PIC S9(9) COMP-5.
   49 MY-FILE-DATA-LENGTH PIC S9(9) COMP-5.
   49 MY-FILE-FILE-OPTIONS PIC S9(9) COMP-5.
   49 MY-FILE-NAME PIC X(255).
```

Soporte de estructura del lenguaje principal en COBOL

El precompilador COBOL soporta declaraciones de elementos de datos de grupos en la sección de declaración de variables del lenguaje principal. Entre otras cosas, esto proporciona un sistema taquigráfico para hacer referencia a un conjunto de elementos de datos elementales en una sentencia de SQL. Por ejemplo, se puede utilizar el siguiente elemento de datos de grupo para acceder a algunas de las columnas de la tabla STAFF de la base de datos SAMPLE:

```
01 staff-record.
    05 staff-id       pic s9(4) comp-5.
    05 staff-name.
        49 l          pic s9(4) comp-5.
        49 d          pic x(9).
    05 staff-info.
        10 staff-dept pic s9(4) comp-5.
        10 staff-job  pic x(5).
```

Los elementos de datos de grupos de la sección de declaración pueden tener, como elementos de datos subordinados, cualquiera de los tipos válidos de variable del lenguaje principal descritos anteriormente. Esto incluye todos los tipos de datos numéricos y de tipo carácter, así como los tipos de objeto grande. Los elementos de datos de grupos se pueden anidar hasta un máximo de 10 niveles. Observe que debe declarar los tipos de caracteres VARCHAR con los elementos subordinados al nivel 49, tal como en el ejemplo anterior. Si no están al nivel 49, VARCHAR se trata como elemento de datos de grupos con dos subordinados y está sujeto a las normas de declaración y utilización de elementos de datos de grupos. En el ejemplo anterior, `staff-info` es un elemento de datos de grupos, mientras que

staff-name es VARCHAR. Se aplica el mismo principio a LONG VARCHAR, VARGRAPHIC y LONG VARGRAPHIC. Puede declarar elementos de datos de grupos a cualquier nivel entre 02 y 49.

Puede utilizar elementos de datos de grupos y los subordinados de los mismos de cuatro maneras:

Método 1.

Se puede hacer referencia al grupo entero como una sola variable del lenguaje principal en una sentencia de SQL:

```
EXEC SQL SELECT id, name, dept, job
  INTO :staff-record
  FROM staff WHERE id = 10 END-EXEC.
```

El precompilador convierte la referencia a staff-record en una lista de todos los elementos subordinados declarados dentro de staff-record, separados por comas. Cada elemento elemental se califica con los nombres de grupo de todos los niveles, a fin de evitar conflictos de denominación con otros elementos. Esto es equivalente al siguiente método.

Método 2.

La segunda manera de utilizar elementos de datos de grupos:

```
EXEC SQL SELECT id, name, dept, job
  INTO
  :staff-record.staff-id,
  :staff-record.staff-name,
  :staff-record.staff-info.staff-dept,
  :staff-record.staff-info.staff-job
  FROM staff WHERE id = 10 END-EXEC.
```

Nota: La referencia a staff-id se califica con su nombre de grupo utilizando el prefijo staff-record., y no staff-id de staff-record como se hace en COBOL puro.

Suponiendo que no existen otras variables del lenguaje principal que tengan los mismos nombres que los subordinados de staff-record, la sentencia anterior también se puede codificar como en el método 3, eliminando la calificación explícita de grupo.

Método 3.

Aquí, se hace referencia a los elementos subordinados de la forma típica de COBOL, sin calificarlos con su elemento de grupo concreto:

```
EXEC SQL SELECT id, name, dept, job
  INTO
  :staff-id,
  :staff-name,
  :staff-dept,
  :staff-job
  FROM staff WHERE id = 10 END-EXEC.
```

Como en COBOL puro, este método resulta aceptable para el precompilador siempre que un elemento subordinado determinado se pueda calificar de forma exclusiva. Por ejemplo, si staff-job aparece más de una vez en un grupo, el precompilador emite un error indicando que se ha producido una referencia ambigua:

```
SQL0088N La variable del lenguaje principal "staff-job" es ambigua.
```

Método 4.

Para resolver la referencia ambigua, puede utilizar una calificación parcial del elemento subordinado, por ejemplo:

```
EXEC SQL SELECT id, name, dept, job
  INTO
  :staff-id,
  :staff-name,
  :staff-info.staff-dept,
  :staff-info.staff-job
  FROM staff WHERE id = 10 END-EXEC.
```

Puesto que una referencia a un solo elemento de grupo, como en el método 1, es equivalente a una lista de sus subordinados, separados por comas, existen casos en que este tipo de referencia conduce a un error. Por ejemplo:

```
EXEC SQL CONNECT TO :staff-record END-EXEC.
```

Aquí, la sentencia CONNECT espera una sola variable del lenguaje principal basada en caracteres. Proporcionando en su lugar el elemento de datos de grupos `staff-record`, la variable del lenguaje principal tiene como resultado el error de precompilación siguiente:

```
SQL0087N  La variable del lenguaje principal "staff-record" es una estructura
          que se utiliza donde no se permiten referencias a estructuras.
```

Otros usos de los elementos de grupos que pueden ocasionar que se produzca un error SQL0087N incluyen: PREPARE, EXECUTE IMMEDIATE, CALL, variables de indicador y referencias a SQLDA. Los grupos que sólo tienen un subordinado están permitidos en ambas situaciones, puesto que se trata de referencias a subordinados individuales, como en los métodos 2, 3 y 4 anteriores.

Tablas de indicadores en COBOL

El precompilador COBOL da soporte a la declaración de tablas de variables de indicador, que es conveniente utilizar con elementos de datos de grupos. Se declaran del modo siguiente:

```
01 <indicator-table-name>.
    05 <indicator-name> pic s9(4) comp-5
                        occurs <table-size> times.
```

Por ejemplo:

```
01 staff-indicator-table.
    05 staff-indicator pic s9(4) comp-5
                       occurs 7 times.
```

Esta tabla de indicadores se puede utilizar eficazmente con el primer formato de referencia de elementos de grupos indicado anteriormente:

```
EXEC SQL SELECT id, name, dept, job
  INTO :staff-record :staff-indicator
  FROM staff WHERE id = 10 END-EXEC.
```

Aquí, el precompilador detecta que `staff-indicator` se ha declarado como tabla de indicadores y la expande en referencias a indicadores individuales cuando procesa la sentencia de SQL. `staff-indicator`(1) se asocia a `staff-id` de `staff-record`, `staff-indicator`(2) se asocia a `staff-name` de `staff-record`, y así sucesivamente.

Nota: Si en la tabla de indicadores existen k entradas de indicador más que subordinados tiene el elemento de datos (por ejemplo, si staff-indicator tiene 10 entradas, haciendo que k=6), se pasan por alto las k entradas de más que se encuentran al final de la tabla de indicadores. Del mismo modo, si hay k entradas de indicador menos que subordinados, los k últimos subordinados del elemento de grupo no tienen asociado ningún indicador. *Tenga en cuenta que puede hacer referencia a elementos individuales en una tabla de indicadores en una sentencia de SQL.*

Conceptos relacionados:

• "Variables de indicador en COBOL" en la página 204

REDEFINES en elementos de datos de grupos COBOL

Cuando declare variables del lenguaje principal, puede utilizar la cláusula REDEFINES. Si declara un miembro de un elemento de datos de grupo con la cláusula REDEFINES y se hace referencia a ese elemento de datos de grupo como un todo en una sentencia de SQL, los elementos subordinados que contienen la cláusula REDEFINES no se expanden. Por ejemplo:

```
01 foo.
 10 a pic s9(4) comp-5.
 10 a1 redefines a pic x(2).
 10 b pic x(10).
```

La referencia a foo en una sentencia de SQL es como sigue:

```
... INTO :foo ...
```

La sentencia anterior es equivalente a:

```
... INTO :foo.a, :foo.b ...
```

Es decir, el elemento subordinado a1, que se declara con la cláusula REDEFINES, no se expande automáticamente en estas situaciones. Si a1 es inequívoco, se puede hacer una referencia explícita a un subordinado que tenga una cláusula REDEFINES en una sentencia de SQL, del modo siguiente:

```
... INTO :foo.a1 ...
```

o

```
... INTO :a1 ...
```

Sección declare de SQL con variables del lenguaje principal para COBOL

A continuación se muestra un ejemplo de sección de declaración de SQL con una variable del lenguaje principal declarada para los tipos de datos SQL soportados.

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
*
  01 age        PIC S9(4) COMP-5.
  01 divis      PIC S9(9) COMP-5.
  01 salary     PIC S9(6)V9(3) COMP-3.
  01 bonus      USAGE IS COMP-1.
  01 wage       USAGE IS COMP-2.
  01 nm         PIC X(5).
  01 varchar.
     49 leng    PIC S9(4) COMP-5.
     49 strg    PIC X(14).
  01 longvchar.
```

```
    49 len     PIC S9(4) COMP-5.
    49 str     PIC X(6027).
  01 MY-CLOB USAGE IS SQL TYPE IS CLOB(1M).
  01 MY-CLOB-LOCATOR USAGE IS SQL TYPE IS CLOB-LOCATOR.
  01 MY-CLOB-FILE USAGE IS SQL TYPE IS CLOB-FILE.
  01 MY-BLOB USAGE IS SQL TYPE IS BLOB(1M).
  01 MY-BLOB-LOCATOR USAGE IS SQL TYPE IS BLOB-LOCATOR.
  01 MY-BLOB-FILE USAGE IS SQL TYPE IS BLOB-FILE.
  01 MY-DBCLOB USAGE IS SQL TYPE IS DBCLOB(1M).
  01 MY-DBCLOB-LOCATOR USAGE IS SQL TYPE IS DBCLOB-LOCATOR.
  01 MY-DBCLOB-FILE USAGE IS SQL TYPE IS DBCLOB-FILE.
  01 MY-PICTURE PIC G(16000) USAGE IS DISPLAY-1.
  01 dt         PIC X(10).
  01 tm         PIC X(8).
  01 tmstmp     PIC X(26).
  01 wage-ind   PIC S9(4) COMP-5.
*
EXEC SQL END DECLARE SECTION END-EXEC.
```

Información relacionada:

- "Tipos de datos de SQL soportados en COBOL" en la página 210

Consideraciones sobre tipos de datos para COBOL

Las secciones siguientes describen cómo se correlacionan los tipos de datos de SQL con tipos de datos de COBOL.

Tipos de datos de SQL soportados en COBOL

Ciertos tipos de datos de COBOL predefinidos corresponden a tipos de columna. Sólo estos tipos de datos de COBOL se pueden declarar como variables del lenguaje principal.

La tabla siguiente muestra el equivalente en COBOL de cada tipo de columna. Cuando el precompilador encuentra una declaración de variable del lenguaje principal, determina el valor del tipo de SQL adecuado. El gestor de bases de datos utiliza este valor para convertir los datos intercambiados entre la aplicación y él mismo.

No se reconoce cada descripción de datos posible para las variables del lenguaje principal. Los elementos de datos de COBOL deben ser coherentes con los descritos en la tabla siguiente. Si utiliza otros elementos de datos, se puede producir un error.

Nota: No existe soporte de variable del lenguaje principal para el tipo de datos DATALINK en ninguno de los lenguajes principales de DB2.

Tabla 15. Tipos de datos de SQL correlacionados con declaraciones de COBOL

Tipo de columna de SQL[1]	Tipo de datos de COBOL	Descripción del tipo de columna de SQL
SMALLINT (500 ó 501)	01 name PIC S9(4) COMP-5.	Entero con signo de 16 bits
INTEGER (496 ó 497)	01 name PIC S9(9) COMP-5.	Entero con signo de 32 bits
BIGINT (492 ó 493)	01 name PIC S9(18) COMP-5.	Entero con signo de 64 bits
DECIMAL(*p*,*s*) (484 ó 485)	01 name PIC S9(*m*)V9(*n*) COMP-3.	Decimal empaquetado

Tabla 15. Tipos de datos de SQL correlacionados con declaraciones de COBOL (continuación)

Tipo de columna de SQL[1]	Tipo de datos de COBOL	Descripción del tipo de columna de SQL
REAL[2] (480 ó 481)	01 name USAGE IS COMP-1.	Coma flotante de precisión simple
DOUBLE[3] (480 ó 481)	01 name USAGE IS COMP-2.	Coma flotante de precisión doble
CHAR(*n*) (452 ó 453)	01 name PIC X(*n*).	Serie de caracteres de longitud fija
VARCHAR(*n*) (448 ó 449)	01 name. 49 length PIC S9(4) COMP-5. 49 name PIC X(*n*). 1<=*n*<=32 672	Serie de caracteres de longitud variable
LONG VARCHAR (456 ó 457)	01 name. 49 length PIC S9(4) COMP-5. 49 data PIC X(*n*). 32 673<=*n*<=32 700	Serie de caracteres de longitud variable larga
CLOB(*n*) (408 ó 409)	01 MY-CLOB USAGE IS SQL TYPE IS CLOB(n). 1<=*n*<=2 147 483 647	Serie de caracteres de longitud variable y objeto grande
Variable de localizador de CLOB[4] (964 ó 965)	01 MY-CLOB-LOCATOR USAGE IS SQL TYPE IS CLOB-LOCATOR.	Identifica entidades CLOB que residen en el servidor
Variable de referencia a archivos CLOB[4] (920 ó 921)	01 MY-CLOB-FILE USAGE IS SQL TYPE IS CLOB-FILE.	Descriptor de un archivo que contiene datos CLOB
BLOB(*n*) (404 ó 405)	01 MY-BLOB USAGE IS SQL TYPE IS BLOB(n). 1<=*n*<=2 147 483 647	Serie binaria de longitud variable y objeto grande
Variable de localizador de BLOB[4] (960 ó 961)	01 MY-BLOB-LOCATOR USAGE IS SQL TYPE IS BLOB-LOCATOR.	Identifica entidades BLOB que residen en el servidor
Variable de referencia a archivos BLOB[4] (916 ó 917)	01 MY-CLOB-FILE USAGE IS SQL TYPE IS CLOB-FILE.	Descriptor de un archivo que contiene datos CLOB
DATE (384 ó 385)	01 identifier PIC X(10).	Serie de caracteres de 10 bytes
TIME (388 ó 389)	01 identifier PIC X(8).	Serie de caracteres de 8 bytes
TIMESTAMP (392 ó 393)	01 identifier PIC X(26).	Serie de caracteres de 26 bytes
Nota: Los tipos de datos siguientes sólo están disponibles en el entorno DBCS.		
GRAPHIC(*n*) (468 ó 469)	01 name PIC G(*n*) DISPLAY-1.	Serie de caracteres de doble byte de longitud fija
VARGRAPHIC(*n*) (464 ó 465)	01 name. 49 length PIC S9(4) COMP-5. 49 name PIC G(*n*) DISPLAY-1. 1<=*n*<=16 336	Serie de caracteres de doble byte de longitud variable con indicador de longitud de serie de 2 bytes
LONG VARGRAPHIC (472 ó 473)	01 name. 49 length PIC S9(4) COMP-5. 49 name PIC G(*n*) DISPLAY-1. 16 337<=*n*<=16 350	Serie de caracteres de doble byte de longitud variable con indicador de longitud de serie de 2 bytes

Tabla 15. Tipos de datos de SQL correlacionados con declaraciones de COBOL (continuación)

Tipo de columna de SQL[1]	Tipo de datos de COBOL	Descripción del tipo de columna de SQL
DBCLOB(*n*) (412 ó 413)	01 MY-DBCLOB USAGE IS SQL TYPE IS DBCLOB(n). 1<=*n*<=1 073 741 823	Serie de caracteres de doble byte de longitud variable y de objeto grande con indicador de longitud de serie de 4 bytes
Variable de localizador de DBCLOB[4] (968 ó 969)	01 MY-DBCLOB-LOCATOR USAGE IS SQL TYPE IS DBCLOB-LOCATOR.	Identifica entidades DBCLOB que residen en el servidor
Variable de referencia a archivos DBCLOB[4] (924 ó 925)	01 MY-DBCLOB-FILE USAGE IS SQL TYPE IS DBCLOB-FILE.	Descriptor de un archivo que contiene datos DBCLOB

Notas:

1. El primer número debajo de **Tipo de columna de SQL** indica que no se proporciona una variable de indicador, y el segundo número indica que sí se proporciona dicha variable. Es necesaria una variable de indicador para indicar los valores NULL o para que contenga la longitud de una serie truncada. Éstos son los valores que aparecerían en el campo SQLTYPE de la SQLDA para estos tipos de datos.
2. FLOAT(*n*) donde 0 < *n* < 25 es un sinónimo de REAL. La diferencia entre REAL y DOUBLE en la SQLDA es el valor de la longitud (4 u 8).
3. Los tipos de SQL siguientes son sinónimos de DOUBLE:
   - FLOAT
   - FLOAT(*n*) donde 24 < *n* < 54
   - DOUBLE PRECISION
4. No es un tipo de columna, sino un tipo de variable del lenguaje principal.

A continuación, se facilitan reglas adicionales para los tipos de datos de COBOL soportados:

- PIC S9 y COMP-3/COMP-5 son necesarios donde se muestran.
- Se puede utilizar el número de nivel 77 en lugar del 01 para todos los tipos de columna, excepto VARCHAR, LONG VARCHAR, VARGRAPHIC, LONG VARGRAPHIC y todos los tipos de variable de LOB.
- Utilice las reglas siguientes al declarar variables del lenguaje principal para los tipos de columna DECIMAL(p,s). Vea el ejemplo siguiente:

  ```
  01 identifier PIC S9(m)V9(n) COMP-3
  ```

  – Utilice V para indicar la coma decimal.
  – Los valores para *n* y *m* deben ser superiores o equivalentes a 1.
  – El valor de *n* + *m* no puede ser superior a 31.
  – El valor para *s* equivale al valor para *n*.
  – El valor para *p* equivale al valor de *n* + *m*.
  – Los factores de repetición *(n)* y *(m)* son opcionales. Todos los ejemplos siguientes resultan válidos:

  ```
  01 identifier PIC S9(3)V COMP-3
  01 identifier PIC SV9(3) COMP-3
  01 identifier PIC S9V COMP-3
  01 identifier PIC SV9 COMP-3
  ```

  – Se puede utilizar PACKED-DECIMAL en lugar de COMP-3.
- Las matrices *no* están soportadas por el precompilador de COBOL.

Conceptos relacionados:

- "Sección declare de SQL con variables del lenguaje principal para COBOL" en la página 209

Tipos de datos BINARY/COMP-4 de COBOL

El precompilador COBOL de DB2® da soporte al uso de los tipos de datos BINARY, COMP y COMP-4 dondequiera que estén permitidas las variables del lenguaje principal y los indicadores, siempre que el compilador COBOL de destino vea (o se pueda hacer que vea) los tipos de datos BINARY, COMP o COMP-4 como equivalentes al tipo de datos COMP-5. En esta publicación, estas variables del lenguaje principal e indicadores se muestran con el tipo COMP-5. Los compiladores de destino soportados por DB2 que tratan COMP, COMP-4, BINARY COMP y COMP-5 como equivalentes son:

- IBM® COBOL Set para AIX®
- Micro Focus COBOL para AIX

FOR BIT DATA en COBOL

Determinadas columnas de bases de datos se pueden declarar FOR BIT DATA. Estas columnas, que generalmente contienen caracteres, se utilizan para contener información binaria. Los tipos de datos CHAR(*n*), VARCHAR, LONG VARCHAR y BLOB son los tipos de variable del lenguaje principal de COBOL que pueden contener datos binarios. Utilice estos tipos de datos cuando trabaje con columnas que tengan el atributo FOR BIT DATA.

Información relacionada:

- "Tipos de datos de SQL soportados en COBOL" en la página 210

Variables SQLSTATE y SQLCODE en COBOL

Cuando se utiliza la opción de precompilación LANGLEVEL con el valor SQL92E, se pueden incluir las dos declaraciones siguientes como variables del lenguaje principal:

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 SQLSTATE PICTURE X(5).
01 SQLCODE  PICTURE S9(9) USAGE COMP.
.
.
.
EXEC SQL END DECLARE SECTION END-EXEC.
```

Si no se especifica ninguna de ellas, se supone la declaración SQLCODE durante el paso de precompilación. `01` también puede ser `77` y `PICTURE` puede ser `PIC`. Observe que, si se utiliza esta opción, no se debe especificar la sentencia INCLUDE SQLCA.

Para las aplicaciones formadas por varios archivos fuente, las declaraciones SQLCODE y SQLSTATE se pueden incluir en cada uno de los archivos fuente, tal como en el ejemplo anterior.

Consideraciones sobre EUC en japonés o chino tradicional y UCS-2 para COBOL

Los datos gráficos enviados desde una aplicación que se ejecuta bajo un juego de códigos eucJp o eucTW, o se conecta a una base de datos UCS-2, se identifican con el identificador de la página de códigos UCS-2. La aplicación debe convertir una serie de caracteres gráficos a UCS-2 antes de enviarla al servidor de bases de datos.

Del mismo modo, los datos gráficos recuperados de una base de datos UCS-2 por una aplicación, o de cualquier base de datos por una aplicación que se ejecute bajo una página de códigos EUC eucJP o eucTW, se codifican utilizando UCS-2. Esto requiere que la aplicación realice internamente una conversión de UCS-2 a la página de códigos de la aplicación, a menos que se vayan a presentar datos UCS-2 al usuario.

La aplicación es responsable de convertir a UCS-2 y desde UCS-2, puesto que esta conversión se debe llevar a cabo antes de copiar los datos al SQLDA o desde esta. DB2 Universal Database no suministra ninguna rutina de conversión que resulte accesible para la aplicación. En cambio, se deben utilizar las llamadas del sistema disponibles desde el sistema operativo. En el caso de una base de datos UCS-2, también puede considerar la posibilidad de utilizar las funciones escalares VARCHAR y VARGRAPHIC.

Conceptos relacionados:

- "Consideraciones sobre los juegos de códigos EUC y UCS-2 de japonés y chino tradicional" en la página 656

Información relacionada:

- "Función escalar VARCHAR" en la publicación *Consulta de SQL, Volumen 1*
- "Función escalar VARGRAPHIC" en la publicación *Consulta de SQL, Volumen 1*

COBOL orientado a objetos

Si utiliza COBOL orientado a objetos, debe tener en cuenta lo siguiente:

- Sólo pueden aparecer sentencias de SQL en el primer programa o clase de una unidad de compilación. Esta restricción se debe a que el precompilador inserta datos de trabajo temporales en la primera sección Working-Storage que encuentra.
- En un programa en COBOL orientado a objetos, cada clase que contenga sentencias de SQL debe tener una sección Working-Storage a nivel de clase, aunque esté vacía. Esta sección se utiliza para almacenar definiciones de datos generadas por el precompilador.

Capítulo 9. Programación en FORTRAN

Consideraciones sobre la programación en FORTRAN 215
Restricciones del lenguaje en FORTRAN 215
Llamada por referencia en FORTRAN 215
Líneas de depuración y de comentario en FORTRAN 216
Consideraciones sobre la precompilación en FORTRAN 216
Acceso a bases de datos de varias hebras en FORTRAN 216
Archivos de entrada y salida para FORTRAN . . 216
Archivos include 216
Archivos include para FORTRAN 216
Archivos include en aplicaciones FORTRAN . . 219
Sentencias de SQL incorporado en FORTRAN . . 219
Variables del lenguaje principal en FORTRAN . . 221
Variables del lenguaje principal en FORTRAN 221
Nombres de variables del lenguaje principal en FORTRAN 221
Declaraciones de variables del lenguaje principal en FORTRAN 222
Sintaxis de las variables numéricas del lenguaje principal en FORTRAN 222
Sintaxis de las variables de tipo carácter del lenguaje principal en FORTRAN 223
Variables de indicador en FORTRAN 224
Sintaxis de las variables del lenguaje principal de objeto grande (LOB) en FORTRAN 225
Sintaxis de las variables del lenguaje principal de localizador de objeto grande (LOB) en FORTRAN 226
Sintaxis de las variables del lenguaje principal de referencia de archivos en FORTRAN . . . 226
Sección declare de SQL con variables del lenguaje principal para FORTRAN 227
Tipos de datos SQL soportados en FORTRAN . . 227
Consideraciones sobre juegos de caracteres de varios bytes en FORTRAN 229
Consideraciones sobre EUC en japonés o chino tradicional y UCS-2 para FORTRAN. 229
Variables SQLSTATE y SQLCODE en FORTRAN 229

Consideraciones sobre la programación en FORTRAN

En las secciones siguientes se explican las consideraciones especiales sobre la programación en el lenguaje principal. Se incluye información sobre las restricciones de lenguaje, los archivos include específicos del lenguaje principal, la incorporación de sentencias de SQL y los tipos de datos soportados para las variables del lenguaje principal.

Nota: Soporte de FORTRAN estabilizado en DB2 Versión 5, y no hay ninguna mejora de FORTRAN planificada para el futuro. Por ejemplo, el precompilador FORTRAN no puede manejar identificadores de objetos SQL, como por ejemplo nombres de tabla, que tengan una longitud superior a 18 bytes. Para utilizar las características incorporadas en DB2 después de la Versión 5, como por ejemplo los nombres de tabla de longitud entre 19 y 128 bytes, debe escribir sus aplicaciones en un lenguaje que no sea FORTRAN.

Restricciones del lenguaje en FORTRAN

Las siguientes secciones describen las restricciones del lenguaje correspondientes a FORTRAN.

Llamada por referencia en FORTRAN

Algunos parámetros de las API requieren direcciones en lugar de valores en las variables de llamada. El gestor de bases de datos proporciona las API GET ADDRESS, DEREFERENCE ADDRESS y COPY MEMORY, que simplifican la posibilidad de que el usuario proporcione estos parámetros.

Información relacionada:

- "sqlgdref - Dereference Address" en la publicación *Administrative API Reference*

- "sqlgaddr - Get Address" en la publicación *Administrative API Reference*
- "sqlgmcpy - Copy Memory" en la publicación *Administrative API Reference*

Líneas de depuración y de comentario en FORTRAN

Algunos compiladores de FORTRAN tratan las líneas que contienen una 'D' o una 'd' en la columna 1 como líneas condicionales. Estas líneas se pueden compilar para su depuración o se pueden tratar como comentarios. El precompilador siempre tratará las líneas que contienen una 'D' o una 'd' en la columna 1 como comentarios.

Consideraciones sobre la precompilación en FORTRAN

Los elementos siguientes afectan al proceso de precompilación:

- El precompilador sólo admite dígitos, blancos y caracteres de tabulación en las columnas 1-5 de las líneas de continuación.
- Las constantes Hollerith no se reciben soporte en archivos fuente `.sqf`.

Acceso a bases de datos de varias hebras en FORTRAN

FORTRAN no da soporte al acceso a bases de datos de varias hebras.

Archivos de entrada y salida para FORTRAN

Por omisión, el archivo de entrada tiene la extensión `.sqf`, pero si se utiliza la opción de precompilación TARGET, el archivo de entrada puede tener la extensión que desee el usuario.

Por omisión, el archivo de salida tiene la extensión `.f` en plataformas basadas en UNIX® y la extensión `.for` en plataformas basadas en Windows®; sin embargo, puede utilizar la opción de precompilación OUTPUT para especificar un nuevo nombre y vía de acceso para el archivo fuente de salida modificado.

Información relacionada:

- "Mandato PRECOMPILE" en la publicación *Consulta de mandatos*

Archivos include

Las siguientes secciones describen los archivos include correspondientes a FORTRAN.

Archivos include para FORTRAN

Los archivos include específicos del lenguaje principal para FORTRAN tienen la extensión `.f` en plataformas basadas en UNIX y la extensión `.for` en plataformas basadas en Windows. Puede utilizar los siguientes archivos include de FORTRAN en las aplicaciones.

SQL (sql.f) Este archivo incluye prototipos específicos del lenguaje para el vinculador, el precompilador y las API de recuperación de mensajes de error. También define constantes del sistema.

SQLAPREP (sqlaprep.f)
Este archivo contiene definiciones necesarias para escribir su propio precompilador.

SQLCA (sqlca_cn.f, sqlca_cs.f)
Este archivo define la estructura del Área de comunicaciones de SQL (SQLCA). El SQLCA contiene variables que utiliza el gestor de bases de datos para proporcionar a una aplicación información de error sobre la ejecución de sentencias de SQL y llamadas a API.

Se proporcionan dos archivos SQLCA para las aplicaciones FORTRAN. El valor por omisión, `sqlca_cs.f`, define la estructura SQLCA en un formato compatible con SQL de IBM. El archivo `sqlca_cn.f`, precompilado con la opción `SQLCA NONE`, define la estructura SQLCA para un mejor rendimiento.

SQLCA_92 (sqlca_92.f)
Este archivo contiene una versión que se ajusta a FIPS SQL92 Entry Level de la estructura Área de comunicaciones de SQL (SQL Communications Area (SQLCA)). Se debe incluir este archivo en lugar de los archivos `sqlca_cn.f` o `sqlca_cs.f` cuando se escriban aplicaciones DB2 que se ajusten al estándar FIPS SQL92 Entry Level. El precompilador de DB2 incluye automáticamente el archivo `sqlca_92.f` cuando la opción LANGLEVEL del precompilador se establece en SQL92E.

SQLCODES (sqlcodes.f)
Este archivo define constantes para el campo SQLCODE de la estructura SQLCA.

SQLDA (sqldact.f)
Este archivo define la estructura del Área de descriptores de SQL (SQLDA). La SQLDA se utiliza para pasar datos entre una aplicación y el gestor de bases de datos.

SQLEAU (sqleau.f)
Este archivo contiene definiciones de constantes y de estructuras necesarias para las API de auditoría de seguridad de DB2. Si utiliza estas API, tiene que incluir este archivo en el programa. Este archivo también contiene definiciones de constantes y de valores de palabras clave para los campos del registro de seguimiento de auditoría. Programas externos o de extracción de seguimiento de auditoría de proveedores pueden utilizar estas definiciones.

SQLENV (sqlenv.f)
Este archivo define llamadas específicas del lenguaje para las API del entorno de bases de datos y las estructuras, constantes y códigos de retorno correspondientes a dichas interfaces.

SQLE819A (sqle819a.f)
Si la página de códigos de la base de datos es 819 (ISO Latin-1), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 500 (EBCDIC internacional) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLE819B (sqle819b.f)
Si la página de códigos de la base de datos es 819 (ISO Latin-1), esta secuencia clasifica series de caracteres que no son FOR BIT

DATA según la clasificación binaria CCSID 037 (EBCDIC inglés de EE.UU.) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLE850A (sqle850a.f)
Si la página de códigos de la base de datos es 850 (ASCII Latin-1), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 500 (EBCDIC internacional) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLE850B (sqle850b.f)
Si la página de códigos de la base de datos es 850 (ASCII Latin-1), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 037 (EBCDIC inglés de EE.UU.) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLE932A (sqle932a.f)
Si la página de códigos de la base de datos es 932 (ASCII japonés), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 5035 (EBCDIC japonés) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLE932B (sqle932b.f)
Si la página de códigos de la base de datos es 932 (ASCII japonés), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 5026 (EBCDIC japonés) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQL1252A (sql1252a.f)
Si la página de códigos de la base de datos es 1252 (Windows Latin-1), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 500 (EBCDIC internacional) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQL1252B (sql1252b.f)
Si la página de códigos de la base de datos es 1252 (Windows Latin-1), esta secuencia clasifica series de caracteres que no son FOR BIT DATA según la clasificación binaria CCSID 037 (EBCDIC inglés de EE.UU.) del sistema principal. La API CREATE DATABASE utiliza este archivo.

SQLMON (sqlmon.f)
Este archivo define llamadas específicas del lenguaje para las API del supervisor del sistema de bases de datos y las estructuras, constantes y códigos de retorno correspondientes a dichas interfaces.

SQLSTATE (sqlstate.f)
Este archivo define constantes correspondientes al campo SQLSTATE de la estructura SQLCA.

SQLUTIL (sqlutil.f)
Este archivo define las llamadas específicas del lenguaje correspondientes a las API de programas de utilidad y las estructuras, constantes y códigos necesarios para dichas interfaces.

Conceptos relacionados:

- "Archivos include en aplicaciones FORTRAN" en la página 219

Archivos include en aplicaciones FORTRAN

Existen dos métodos para incluir archivos: la sentencia EXEC SQL INCLUDE y la sentencia INCLUDE de FORTRAN. El precompilador pasará por alto las sentencias INCLUDE de FORTRAN y sólo procesará los archivos incluidos mediante la sentencia EXEC SQL.

Para localizar el archivo INCLUDE, el precompilador FORTRAN de DB2® busca en primer lugar en el directorio actual y, a continuación, en los directorios especificados por la variable de entorno DB2INCLUDE. Considere los ejemplos siguientes:

- `EXEC SQL INCLUDE payroll`

  Si el archivo especificado en la sentencia INCLUDE no está encerrado entre comillas, como en el ejemplo anterior, el precompilador busca sucesivamente `payroll.sqf` y `payroll.f` (`payroll.for` en las plataformas basadas en Windows®) en cada uno de los directorios que examina.

- `EXEC SQL INCLUDE 'pay/payroll.f'`

  Si el nombre de archivo está encerrado entre comillas, tal como en el caso anterior, no se añade ninguna extensión al nombre. (Para plataformas basadas en Windows, el archivo se especificaría como `'pay\payroll.for'`.)

  Si el nombre de archivo entrecomillado no contiene una vía de acceso absoluta, se utiliza el contenido de DB2INCLUDE para buscar el archivo, añadiéndole como prefijo la vía de acceso especificada en el nombre del archivo de INCLUDE. Por ejemplo, en DB2 para plataformas basadas en UNIX®, si DB2INCLUDE tiene el valor '`/disk2:myfiles/fortran`', el precompilador busca sucesivamente '`./pay/payroll.f`', '`/disk2/pay/payroll.f`' y finalmente '`./myfiles/cobol/pay/payroll.f`'. En los mensajes del precompilador se muestra la vía de acceso en que se encuentra realmente el archivo. En plataformas basadas en Windows, sustituya las barras inclinadas invertidas (\) por barras inclinadas y sustituya `'for'` por la extensión `'f'` en el ejemplo anterior.

Nota: El procesador de línea de mandatos de DB2 coloca en antememoria el valor de DB2INCLUDE. Para cambiar el valor de DB2INCLUDE después de haber emitido mandatos del CLP, entre el mandato TERMINATE, luego vuelva a conectar con la base de datos y realice una precompilación del modo habitual.

Conceptos relacionados:

- "DB2 registry and environment variables" en la publicación *Administration Guide: Performance*

Información relacionada:

- "Archivos include para FORTRAN" en la página 216

Sentencias de SQL incorporado en FORTRAN

Las sentencias de SQL incorporado constan de los tres elementos siguientes:

Elemento	Sintaxis correcta en FORTRAN
Palabra clave	EXEC SQL

Serie de la sentencia	Cualquier sentencia de SQL válida con blancos como delimitadores
Terminador de la sentencia	Fin de la línea fuente.

El fin de la línea fuente sirve como terminador de sentencia. Si se continúa la línea, el terminador de sentencia es el fin de la última línea de continuación.

Por ejemplo:

```
EXEC SQL SELECT COL INTO :hostvar FROM TABLE
```

Se aplican las normas siguientes a las sentencias de SQL incorporado:

- Codifique las sentencias de SQL únicamente entre las columnas 7 y 72.
- Utilice comentarios de FORTRAN de línea completa, o comentarios de SQL, pero no utilice el carácter '!' de comentario de fin de línea de FORTRAN en las sentencias de SQL. Este carácter de comentario se puede utilizar en cualquier otra parte, incluso en las declaraciones de variables del lenguaje principal.
- Utilice blancos como delimitadores cuando codifique sentencias de SQL incorporado, aunque las sentencias de FORTRAN no requieren blancos como delimitadores.
- Utilice únicamente una sentencia de SQL para cada línea fuente en FORTRAN. Para las sentencias que necesitan más de una línea fuente, se aplican las normas habituales de continuación de FORTRAN. No divida el par de palabras clave EXEC SQL en distintas líneas.
- Están permitidos los comentarios de SQL en cualquier línea que forme parte de una sentencia de SQL incorporado. Estos comentarios no están permitidos en las sentencias que se ejecutan de forma dinámica. El formato de un comentario de SQL consiste en un guión doble (--) seguido de una serie compuesta por cero o más caracteres y terminada por un fin de línea.
- Los comentarios FORTRAN están permitidos *casi* en cualquier lugar de una sentencia de SQL incorporada. Las excepciones son:
  - No se permiten comentarios entre EXEC y SQL.
  - No se permiten comentarios en las sentencias que se ejecutan dinámicamente.
  - La ampliación de utilizar ! para codificar un comentario FORTRAN al final de una línea no recibe soporte dentro de una sentencia de SQL incorporado.
- Utilice una notación exponencial cuando especifique una constante real en las sentencias de SQL. El gestor de bases de datos interpreta una serie de dígitos con una coma decimal en una sentencia de SQL como una constante decimal, no como una constante real.
- Los números de sentencia no son válidos en las sentencias de SQL que preceden a la primera sentencia FORTRAN ejecutable. Si una sentencia de SQL tiene asociado un número de sentencia, el precompilador genera una sentencia CONTINUE etiquetada que precede directamente a la sentencia de SQL.
- Cuando haga referencia a variables del lenguaje principal dentro de una sentencia de SQL, utilícelas exactamente tal como se han declarado.
- La sustitución de caracteres de espacio en blanco, como caracteres de fin de línea y de tabulación, se produce del siguiente modo:
  - Cuando aparecen fuera de las comillas (pero dentro de sentencias de SQL), los caracteres de fin de línea y tabuladores se sustituyen por un solo espacio.
  - Si aparecen dentro de las comillas, los caracteres de fin de línea desaparecen, siempre que se continúe correctamente la serie para un programa FORTRAN. Los tabuladores no se modifican.

Observe que los caracteres reales utilizados como fin de línea y tabulador varían en función de la plataforma. Por ejemplo, las plataformas basadas en Windows® utilizan el retorno de carro/salto de línea como fin de línea, mientras que las plataformas basadas en UNIX® utilizan simplemente un salto de línea.

Información relacionada:

- Apéndice A, "Sentencias de SQL soportadas", en la página 733

Variables del lenguaje principal en FORTRAN

Las secciones siguientes describen cómo declarar y utilizar variables del lenguaje principal en programas FORTRAN.

Variables del lenguaje principal en FORTRAN

Las variables del lenguaje principal son variables del lenguaje FORTRAN a las que se hace referencia en sentencias de SQL. Permiten que una aplicación pase datos de entrada al gestor de bases de datos y reciba datos de salida de éste. Una vez que se ha precompilado la aplicación, el compilador utiliza las variables del lenguaje principal como cualquier otra variable de FORTRAN.

Conceptos relacionados:

- "Nombres de variables del lenguaje principal en FORTRAN" en la página 221
- "Declaraciones de variables del lenguaje principal en FORTRAN" en la página 222
- "Variables de indicador en FORTRAN" en la página 224

Información relacionada:

- "Sintaxis de las variables numéricas del lenguaje principal en FORTRAN" en la página 222
- "Sintaxis de las variables de tipo carácter del lenguaje principal en FORTRAN" en la página 223
- "Sintaxis de las variables del lenguaje principal de objeto grande (LOB) en FORTRAN" en la página 225
- "Sintaxis de las variables del lenguaje principal de localizador de objeto grande (LOB) en FORTRAN" en la página 226
- "Sintaxis de las variables del lenguaje principal de referencia de archivos en FORTRAN" en la página 226

Nombres de variables del lenguaje principal en FORTRAN

El precompilador SQL identifica las variables del lenguaje principal por el nombre declarado para éstas. Se aplican las sugerencias siguientes:

- Especifique nombres de variables con una longitud máxima de 255 caracteres.
- Empiece los nombres de variables con prefijos que no sean `SQL`, `sql`, `DB2` ni `db2`, que están reservados para uso del sistema.

Conceptos relacionados:

- "Declaraciones de variables del lenguaje principal en FORTRAN" en la página 222

Información relacionada:

- "Sintaxis de las variables numéricas del lenguaje principal en FORTRAN" en la página 222
- "Sintaxis de las variables de tipo carácter del lenguaje principal en FORTRAN" en la página 223
- "Sintaxis de las variables del lenguaje principal de objeto grande (LOB) en FORTRAN" en la página 225
- "Sintaxis de las variables del lenguaje principal de localizador de objeto grande (LOB) en FORTRAN" en la página 226
- "Sintaxis de las variables del lenguaje principal de referencia de archivos en FORTRAN" en la página 226

Declaraciones de variables del lenguaje principal en FORTRAN

Se debe utilizar una sección de declaración de SQL para identificar las declaraciones de variables del lenguaje principal. Esto alerta al precompilador acerca de las variables del lenguaje principal a las que se puede hacer referencia en sentencias de SQL posteriores.

El precompilador FORTRAN sólo reconoce un subconjunto de declaraciones de FORTRAN válidas como declaraciones de variables del lenguaje principal válidas. Estas declaraciones definen variables numéricas o de tipo carácter. Una variable numérica del lenguaje principal se puede utilizar como variable de entrada o de salida para cualquier valor numérico de entrada o salida de SQL. Una variable del lenguaje principal de tipo carácter se puede utilizar como variable de entrada o de salida para cualquier valor de tipo carácter, de fecha, de hora o de indicación de la hora, de entrada o salida de SQL. El programador se tiene que asegurar de que las variables de salida sean lo suficientemente largas como para contener los valores que van a recibir.

Tareas relacionadas:

- "Declaración de variables del lenguaje principal de tipo estructurado" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*

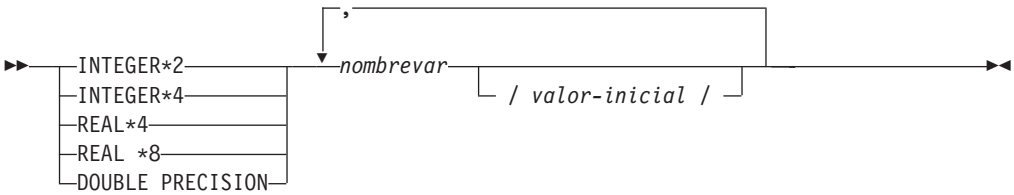
Información relacionada:

- "Sintaxis de las variables numéricas del lenguaje principal en FORTRAN" en la página 222
- "Sintaxis de las variables de tipo carácter del lenguaje principal en FORTRAN" en la página 223
- "Sintaxis de las variables del lenguaje principal de objeto grande (LOB) en FORTRAN" en la página 225
- "Sintaxis de las variables del lenguaje principal de localizador de objeto grande (LOB) en FORTRAN" en la página 226
- "Sintaxis de las variables del lenguaje principal de referencia de archivos en FORTRAN" en la página 226

Sintaxis de las variables numéricas del lenguaje principal en FORTRAN

A continuación se muestra la sintaxis de las variables numéricas del lenguaje principal en FORTRAN.

Sintaxis de las variables numéricas del lenguaje principal en FORTRAN



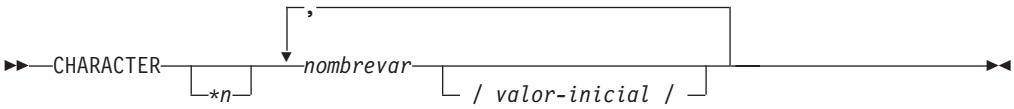
Consideraciones sobre las variables numéricas del lenguaje principal:

1. REAL*8 y DOUBLE PRECISION son equivalentes.
2. Utilice una E en lugar de una D como indicador de exponente para las constantes REAL*8.

Sintaxis de las variables de tipo carácter del lenguaje principal en FORTRAN

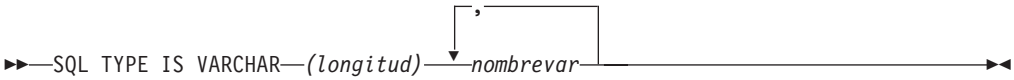
A continuación se muestra la sintaxis de las variables de tipo carácter de longitud fija del lenguaje principal.

Sintaxis de las variables del lenguaje principal de tipo carácter en FORTRAN: Longitud fija



A continuación se muestra la sintaxis de las variables de tipo carácter de longitud variable del lenguaje principal.

Longitud variable



Consideraciones sobre las variables del lenguaje principal de tipo carácter:

1. **n* tiene un valor máximo de 254.
2. Si la longitud está entre 1 y 32 672, ambos inclusive, la variable del lenguaje principal es de tipo VARCHAR(SQLTYPE 448).
3. Si la longitud está entre 32 673 y 32 700, ambos inclusive, la variable del lenguaje principal es de tipo LONG VARCHAR(SQLTYPE 456).
4. No está permitida la inicialización de variables del lenguaje principal VARCHAR y LONG VARCHAR dentro de la declaración.

Ejemplo de VARCHAR:

La declaración:

```
sql type is varchar(1000) my_varchar
```

Tiene como resultado la generación de la estructura siguiente:

```
character    my_varchar(1000+2)
integer*2    my_varchar_length
character    my_varchar_data(1000)
equivalence( my_varchar(1),
+            my_varchar_length )
equivalence( my_varchar(3),
+            my_varchar_data )
```

La aplicación puede manipular tanto my_varchar_length como my_varchar_data; por ejemplo, para establecer o examinar el contenido de la variable del lenguaje principal. El nombre base (en este caso, my_varchar), se utiliza en las sentencias de SQL para hacer referencia a la VARCHAR como un todo.

Ejemplo de LONG VARCHAR:

La declaración:

```
sql type is varchar(10000) my_lvarchar
```

Tiene como resultado la generación de la estructura siguiente:

```
character    my_lvarchar(10000+2)
integer*2    my_lvarchar_length
character    my_lvarchar_data(10000)
equivalence( my_lvarchar(1),
+            my_lvarchar_length )
equivalence( my_lvarchar(3),
+            my_lvarchar_data )
```

La aplicación puede manipular tanto my_lvarchar_length como my_lvarchar_data; por ejemplo, para establecer o examinar el contenido de la variable del lenguaje principal. El nombre base (en este caso, my_lvarchar) se utiliza en las sentencias de SQL para hacer referencia a la LONG VARCHAR como un todo.

Nota: En una sentencia CONNECT, como por ejemplo la que se muestra a continuación, a las variables del lenguaje principal de serie de caracteres en FORTRAN dbname y userid se les eliminarán los blancos de cola antes del proceso.

```
EXEC SQL CONNECT TO :dbname USER :userid USING :passwd
```

No obstante, puesto que los blancos pueden ser significativos en las contraseñas, debe declarar las variables del lenguaje principal para contraseñas como VARCHAR, y hacer que el campo de longitud establecido refleje la longitud real de la contraseña:

```
EXEC SQL BEGIN DECLARE SECTION
  character*8 dbname, userid
  sql type is varchar(18) passwd
EXEC SQL END DECLARE SECTION
character*18 passwd_string
equivalence(passwd_data,passwd_string)
dbname = 'sample'
userid = 'userid'
passwd_length= 8
passwd_string = 'password'
EXEC SQL CONNECT TO :dbname USER :userid USING :passwd
```

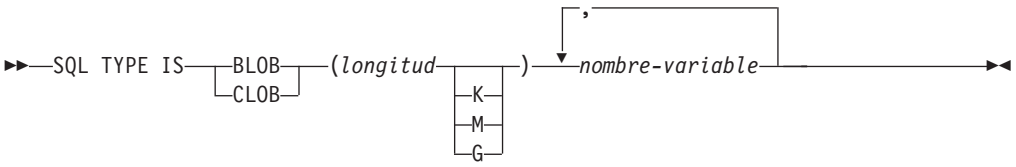
Variables de indicador en FORTRAN

Las variables de indicador se deben declarar con tipo de datos INTEGER*2.

Sintaxis de las variables del lenguaje principal de objeto grande (LOB) en FORTRAN

A continuación se muestra la sintaxis para declarar variables del lenguaje principal de objeto grande (LOB) en FORTRAN.

Sintaxis de las variables del lenguaje principal de objeto grande (LOB) en FORTRAN



Consideraciones sobre las variables del lenguaje principal LOB:

1. Los tipos GRAPHIC no se soportan en FORTRAN.
2. SQL TYPE IS, BLOB, CLOB, K, M, G se pueden especificar en mayúsculas, en minúsculas o en una mezcla de ambas.
3. Para BLOB y CLOB 1 <= longitud-lob <= 2 147 483 647.
4. No se permite la inicialización de un LOB dentro de una declaración LOB.
5. El nombre de la variable del lenguaje principal es el prefijo de 'longitud? y 'datos' en el código generado por el precompilador.

Ejemplo de BLOB:

La declaración:

```
sql type is blob(2m) my_blob
```

Tiene como resultado la generación de la estructura siguiente:

```
 character    my_blob(2097152+4)
 integer*4    my_blob_length
 character    my_blob_data(2097152)
 equivalence( my_blob(1),
+             my_blob_length )
 equivalence( my_blob(5),
+             my_blob_data )
```

Ejemplo de CLOB:

La declaración:

```
sql type is clob(125m) my_clob
```

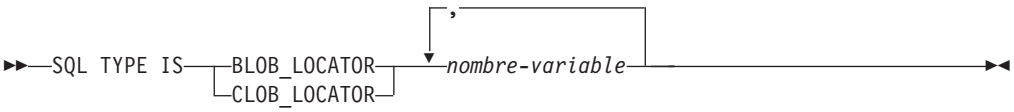
Tiene como resultado la generación de la estructura siguiente:

```
 character    my_clob(131072000+4)
 integer*4    my_clob_length
 character    my_clob_data(131072000)
 equivalence( my_clob(1),
+             my_clob_length )
 equivalence( my_clob(5),
+             my_clob_data )
```

Sintaxis de las variables del lenguaje principal de localizador de objeto grande (LOB) en FORTRAN

A continuación se muestra la sintaxis para declarar variables del lenguaje principal de localizador de objeto grande (LOB) en FORTRAN.

Sintaxis de las variables del lenguaje principal de localizador de objeto grande (LOB) en FORTRAN



Consideraciones sobre las variables del lenguaje principal de localizador de LOB:

1. Los tipos GRAPHIC no se soportan en FORTRAN.
2. SQL TYPE IS, BLOB_LOCATOR, CLOB_LOCATOR se pueden especificar en mayúsculas, en minúsculas o en una mezcla de ambas.
3. No se permite la inicialización de localizadores.

Ejemplo de localizador de CLOB (El localizador de BLOB es parecido):

La declaración:

```
SQL TYPE IS CLOB_LOCATOR my_locator
```

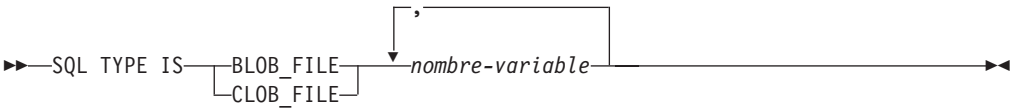
Tiene como resultado la generación de la declaración siguiente:

```
integer*4 my_locator
```

Sintaxis de las variables del lenguaje principal de referencia de archivos en FORTRAN

A continuación se muestra la sintaxis para declarar variables del lenguaje principal de referencia de archivos en FORTRAN.

Sintaxis de las variables del lenguaje principal de referencia de archivos en FORTRAN



Consideraciones sobre las variables del lenguaje principal de referencia de archivos:

1. Los tipos GRAPHIC no se soportan en FORTRAN.
2. SQL TYPE IS, BLOB-FILE, CLOB-FILE se pueden especificar en mayúsculas, en minúsculas o en una mezcla de ambas.

Ejemplo de una variable de referencia de archivos BLOB (La variable de referencia de archivos CLOB es parecida):

```
SQL TYPE IS BLOB_FILE my_file
```

Tiene como resultado la generación de la declaración siguiente:

```
 character     my_file(267)
 integer*4     my_file_name_length
 integer*4     my_file_data_length
 integer*4     my_file_file_options
 character*255 my_file_name
 equivalence(  my_file(1),
+              my_file_name_length )
 equivalence(  my_file(5),
+              my_file_data_length )
 equivalence(  my_file(9),
+              my_file_file_options )
 equivalence(  my_file(13),
+              my_file_name )
```

Sección declare de SQL con variables del lenguaje principal para FORTRAN

A continuación se muestra un ejemplo de sección de declaración de SQL con una variable del lenguaje principal declarada para cada tipo de datos soportado:

```
EXEC SQL BEGIN DECLARE SECTION
  INTEGER*2     AGE  /26/
  INTEGER*4     DEPT
  REAL*4        BONUS
  REAL*8        SALARY
  CHARACTER     MI
  CHARACTER*112 ADDRESS
  SQL TYPE IS VARCHAR (512) DESCRIPTION
  SQL TYPE IS VARCHAR (32000) COMMENTS
  SQL TYPE IS CLOB (1M) CHAPTER
  SQL TYPE IS CLOB_LOCATOR CHAPLOC
  SQL TYPE IS CLOB_FILE  CHAPFL
  SQL TYPE IS BLOB (1M) VIDEO
  SQL TYPE IS BLOB_LOCATOR VIDLOC
  SQL TYPE IS BLOB_FILE VIDFL
  CHARACTER*10  DATE
  CHARACTER*8   TIME
  CHARACTER*26  TIMESTAMP
  INTEGER*2     WAGE_IND
EXEC SQL END DECLARE SECTION
```

Información relacionada:

- "Tipos de datos SQL soportados en FORTRAN" en la página 227

Tipos de datos SQL soportados en FORTRAN

Determinados tipos de datos de FORTRAN predefinidos corresponden a tipos de columna del gestor de bases de datos. Sólo se pueden declarar como variables del lenguaje principal estos tipos de datos de FORTRAN.

La tabla siguiente muestra el equivalente en FORTRAN de cada tipo de columna. Cuando el precompilador encuentra una declaración de una variable del lenguaje principal, determina el valor del tipo de SQL apropiado. El gestor de bases de datos utiliza este valor para convertir los datos intercambiados entre la aplicación y él mismo.

Nota: No existe soporte de variables del lenguaje principal para el tipo de datos DATALINK en ninguno de los lenguajes principales de DB2.

Tabla 16. Tipos de datos SQL correlacionados con declaraciones FORTRAN

Tipo de columna SQL[1]	Tipo de datos FORTRAN	Descripción del tipo de columna SQL
SMALLINT (500 ó 501)	INTEGER*2	Entero con signo de 16 bits
INTEGER (496 ó 497)	INTEGER*4	Entero con signo de 32 bits
REAL[2] (480 ó 481)	REAL*4	Coma flotante de precisión simple
DOUBLE[3] (480 ó 481)	REAL*8	Coma flotante de precisión doble
DECIMAL(*p*,*s*)(484 ó 485)	No existe un equivalente exacto; utilice REAL*8	Decimal empaquetado
CHAR(*n*) (452 ó 453)	CHARACTER**n*	Serie de caracteres de longitud fija con longitud *n*, donde *n* va de 1 a 254
VARCHAR(*n*) (448 ó 449)	SQL TYPE IS VARCHAR(*n*), donde *n* va de 1 a 32 672	Serie de caracteres de longitud variable
LONG VARCHAR (456 ó 457)	SQL TYPE IS VARCHAR(*n*), donde *n* va de 32 673 a 32 700	Serie de caracteres de longitud variable larga
CLOB(*n*) (408 ó 409)	SQL TYPE IS CLOB (n) donde *n* va de 1 a 2 147 483 647	Serie de caracteres de longitud variable y objeto grande
Variable de localizador CLOB[4] (964 ó 965)	SQL TYPE IS CLOB_LOCATOR	Identifica las entidades CLOB que residen en el servidor
Variable de referencia de archivo CLOB[4] (920 ó 921)	SQL TYPE IS CLOB_FILE	Descriptor de archivo que contiene datos CLOB
BLOB(*n*) (404 ó 405)	SQL TYPE IS BLOB (n) donde *n* va de 1 a 2 147 483 647	Serie binaria de longitud variable y objeto grande
Variable de localizador BLOB[4] (960 ó 961)	SQL TYPE IS BLOB_LOCATOR	Identifica las entidades BLOB del servidor
Variable de referencia de archivo BLOB[4] (916 ó 917)	SQL TYPE IS BLOB_FILE	Descriptor del archivo que contiene datos BLOB
DATE (384 ó 385)	CHARACTER*10	Serie de caracteres de 10 bytes
TIME (388 ó 389)	CHARACTER*8	Serie de caracteres de 8 bytes
TIMESTAMP (392 ó 393)	CHARACTER*26	Serie de caracteres de 26 bytes

Notas:

1. El primer número que se encuentra bajo **Tipo de columna SQL** indica que no se proporciona una variable de indicador, y el segundo número indica que se proporciona una variable de indicador. Se necesita una variable de indicador para indicar los valores nulos (NULL) o para contener la longitud de una serie truncada. Éstos son los valores que aparecerán en el campo SQLTYPE de la SQLDA para estos tipos de datos.
2. FLOAT(*n*) donde 0 < *n* < 25 es un sinónimo de REAL. La diferencia entre REAL y DOUBLE en la SQLDA es el valor de la longitud (4 u 8).
3. Los tipos SQL siguientes son sinónimos de DOUBLE:
   - FLOAT
   - FLOAT(*n*) donde 24 < *n* < 54 es
   - DOUBLE PRECISION
4. Éste no es un tipo de columna, sino un tipo de variable del lenguaje principal.

A continuación mostramos una norma adicional para los tipos de datos FORTRAN soportados:

- Puede definir sentencias de SQL dinámicas que contengan más de 254 caracteres utilizando las variables del lenguaje principal VARCHAR, LONG VARCHAR o CLOB.

Conceptos relacionados:

- "Sección declare de SQL con variables del lenguaje principal para FORTRAN" en la página 227

Consideraciones sobre juegos de caracteres de varios bytes en FORTRAN

En FORTRAN no se da soporte a ningún tipo de datos de variables del lenguaje principal de gráficos (de varios bytes). Sólo se da soporte a variables del lenguaje principal de tipo carácter mixtas mediante el tipo de datos `character`. Es posible crear una SQLDA de usuario que contenga datos gráficos.

Consideraciones sobre EUC en japonés o chino tradicional y UCS-2 para FORTRAN

Los datos gráficos enviados desde una aplicación que se ejecuta bajo un juego de códigos eucJp o eucTW, o se conecta a una base de datos UCS-2, se identifican con el identificador de la página de códigos UCS-2. La aplicación debe convertir una serie de caracteres gráficos a UCS-2 antes de enviarla al servidor de bases de datos. Del mismo modo, los datos gráficos recuperados de una base de datos UCS-2 por una aplicación, o de cualquier base de datos por una aplicación que se ejecute bajo una página de códigos EUC eucJP o eucTW, se codifican utilizando UCS-2. Esto requiere que la aplicación realice internamente una conversión de UCS-2 a la página de códigos de la aplicación, a menos que se vayan a presentar datos UCS-2 al usuario.

La aplicación es responsable de convertir a UCS-2 y desde UCS-2, puesto que esta conversión se debe llevar a cabo antes de copiar los datos al SQLDA o desde esta. DB2 Universal Database no suministra ninguna rutina de conversión que resulte accesible para la aplicación. En cambio, se deben utilizar las llamadas del sistema disponibles desde el sistema operativo. En el caso de una base de datos UCS-2, también puede considerar la posibilidad de utilizar las funciones escalares VARCHAR y VARGRAPHIC.

Conceptos relacionados:

- "Consideraciones sobre los juegos de códigos EUC y UCS-2 de japonés y chino tradicional" en la página 656

Información relacionada:

- "Función escalar VARCHAR" en la publicación *Consulta de SQL, Volumen 1*
- "Función escalar VARGRAPHIC" en la publicación *Consulta de SQL, Volumen 1*

Variables SQLSTATE y SQLCODE en FORTRAN

Cuando se utiliza la opción de precompilación LANGLEVEL con el valor SQL92E, se pueden incluir las dos declaraciones siguientes como variables del lenguaje principal:

```
EXEC SQL BEGIN DECLARE SECTION;
  CHARACTER*5 SQLSTATE
  INTEGER     SQLCOD
```

```
    .
    .
    .
EXEC SQL END DECLARE SECTION
```

Si no es especifica ninguna de ellas, se supone la declaración SQLCOD durante el paso de precompilación. La variable denominada SQLSTATE también puede ser SQLSTA. Observe que, si se utiliza esta opción, no se debe especificar la sentencia INCLUDE SQLCA.

Para aplicaciones que contienen varios archivos fuente, se pueden incluir las declaraciones de SQLCOD y SQLSTATE en cada archivo fuente, tal como se ha mostrado anteriormente.

Información relacionada:

- "Mandato PRECOMPILE" en la publicación *Consulta de mandatos*

Parte 3. ADO.NET, OLE DB y ODBC

Capítulo 10. DB2 .NET Data Provider

Visión general de DB2 .NET Data Provider . . . 233
Requisitos del sistema DB2 .NET Data Provider 233
Programación de aplicaciones para utilizar DB2 .NET Data Provider 234
Conexión a una base de datos desde una aplicación utilizando DB2 .NET Data Provider . 234
Ejecución de sentencias de SQL desde una aplicación utilizando DB2 .NET Data Provider . 234
Lectura de conjuntos de resultados de una aplicación utilizando DB2 .NET Data Provider . 235
Invocación de procedimientos almacenados desde una aplicación utilizando DB2 .NET Data Provider 236
Tipos de datos SQL soportados para DB2 .NET Data Provider 238

Visión general de DB2 .NET Data Provider

DB2® .NET Data Provider es una ampliación de la interfaz ADO.NET que permite que las aplicaciones .NET accedan a una base de datos DB2 mediante una conexión segura, ejecuten mandatos y obtengan conjuntos de resultados.

Junto con DB2 .NET Data Provider se incluye documentación de consulta, que muestra información detallada sobre todos los objetos de DB2 .NET Data Provider y sus miembros. Durante el proceso de instalación de DB2, esta documentación se registra con Microsoft® Visual Studio .NET. Para ver la documentación de DB2 .NET Data Provider desde Microsoft Visual Studio .NET, seleccione la opción de menú `Ayuda` y `Contenido`. Cuando se abra el visor de ayuda, realice un filtrado por *Ayuda de IBM® DB2 .NET Data Provider*.

Requisitos del sistema DB2 .NET Data Provider

DB2® .NET Data Provider le permite que sus aplicaciones .NET accedan a los siguientes sistemas de gestión de bases de datos:

- DB2 Universal Database™ Versión 8 para sistemas Windows®, UNIX® y Linux
- DB2 Universal Database Versión 6 (o posterior) para OS/390® y z/OS™, a través de DB2 Connect™
- DB2 Universal Database Versión 5, Release 1 (o posterior) para AS/400® e iSeries™, a través de DB2 Connect
- DB2 Universal Database Versión 7.3 (o posterior) para VSE y VM, a través de DB2 Connect

Antes de utilizar el programa de instalación de DB2 para instalar DB2 .NET Data Provider, debe tener ya instalado en el sistema .NET Framework (Versión 1.0 o Versión 1.1). Si .NET Framework no está instalado, el programa de instalación de DB2 no instalará DB2 .NET Data Provider.

Para DB2 Universal Database para AS/400 e iSeries, es necesario el siguiente arreglo de programa en el servidor: APAR ii13348.

Solo se pueden utilizar .NET Framework Versión 1.1 y Visual Studio .NET 2003 con los servidores DB2 para VSE y VM, y los servidores DB2 para iSeries. Los productos .NET Framework Versión 1.0 y Visual Studio .NET 2002 no se pueden utilizar con esos servidores.

Programación de aplicaciones para utilizar DB2 .NET Data Provider

Las secciones siguientes describen los pasos principales para programar una aplicación .NET para acceder o manejar datos de una base de datos DB2. Se proporcionan ejemplos en C# y Visual Basic .NET para mostrar cada paso.

Conexión a una base de datos desde una aplicación utilizando DB2 .NET Data Provider

Cuando se utiliza DB2 .NET Data Provider, se establece una conexión con una base de datos a través de la clase DB2Connection. Primero, el usuario debe crear una serie de caracteres para contener los parámetros de conexión.

Son ejemplos de series de conexión:

- ```
  String connectString = "Database=SAMPLE";
  // Cuando se utiliza, intenta conectar con la base de datos SAMPLE.
  ```
- ```
  String connectString = "Server=srv:50000;Database=SAMPLE;UID=db2adm;PWD=ab1cd";
  // Cuando se utiliza, intenta conectar con la base de datos SAMPLE del servidor
  // 'srv' a través del puerto 50000 y utilizando 'db2adm' y 'ab1cd'
  // como ID de usuario y contraseña, respectivamente.
  ```

Para crear la conexión de base de datos, pase la serie de conexión connectString al constructor de DB2Connection. Luego utilice el método Open del objeto DB2Connection para conectar formalmente con la base de datos especificada en connectString.

Conexión a una base de datos en C#:

```
String connectString = "Database=SAMPLE";
DB2Connection conn = new DB2Connection(connectString);
conn.Open();
return conn;
```

Conexión a una base de datos en Visual Basic .NET:

```
Dim connectString As String = "Database=SAMPLE"
Dim conn As DB2Connection = new DB2Connection(connectString)
conn.Open()
Return conn
```

Ejecución de sentencias de SQL desde una aplicación utilizando DB2 .NET Data Provider

Cuando se utiliza DB2 .NET Data Provider, la sentencias de SQL se ejecutan a través de la clase DB2Command utilizando sus métodos ExecuteReader() y ExecuteNonQuery(), y sus propiedades CommandText, CommandType y Transaction. Para las sentencias de SQL que generan datos de salida, se debe utilizar el método ExecuteReader() y sus resultados se pueden recuperar de un objeto DB2DataReader. Para todas las demás sentencias de SQL, se debe utilizar el método ExecuteNonQuery(). La propiedad Transaction del objeto DB2Command se debe inicializar para un objeto DB2Transaction. El objeto DB2Transaction es el encargado de retrotraer y confirmar las transacciones de base de datos.

Ejecución de una sentencia UPDATE en C#:

```
// se supone la existencia de una conexión DB2Connection
DB2Command cmd = conn.CreateCommand();
DB2Transaction trans = conn.BeginTransaction();
cmd.Transaction = trans;
cmd.CommandText = "UPDATE staff " +
                  "  SET salary = (SELECT MIN(salary) " +
```

```
        "  FROM staff " +
    "                   WHERE id >= 310) " +
                    "  WHERE id = 310";
cmd.ExecuteNonQuery();
```

Ejecución de una sentencia UPDATE en Visual Basic .NET:

```
' se supone la existencia de una conexión DB2Connection
DB2Command cmd = conn.CreateCommand();
DB2Transaction trans = conn.BeginTransaction();
cmd.Transaction = trans;
cmd.CommandText = "UPDATE staff " +
                  "  SET salary = (SELECT MIN(salary) " +
        "  FROM staff " +
                  "                   WHERE id >= 310) " +
                  "  WHERE id = 310";
cmd.ExecuteNonQuery();
```

Ejecución de una sentencia SELECT en C#:

```
// se supone la existencia de una conexión DB2Connection
DB2Command cmd = conn.CreateCommand();
DB2Transaction trans = conn.BeginTransaction();
cmd.Transaction = trans;
cmd.CommandText = "SELECT deptnumb, location " +
                  "  FROM org " +
                  "  WHERE deptnumb < 25";
DB2DataReader reader = cmd.ExecuteReader();
```

Ejecución de una sentencia SELECT en Visual Basic .NET:

```
' se supone la existencia de una conexión DB2Connection
Dim cmd As DB2Command = conn.CreateCommand()
Dim trans As DB2Transaction = conn.BeginTransaction()
cmd.Transaction = trans
cmd.CommandText = "UPDATE staff " +
                  "  SET salary = (SELECT MIN(salary) " +
        "  FROM staff " +
                  "                   WHERE id >= 310) " +
                  "  WHERE id = 310"
cmd.ExecuteNonQuery()
```

Una vez que su aplicación ha efectuado una transacción de base de datos, debe retrotraerla o confirmarla. Esto se realiza mediante los métodos `Commit()` y `Rollback()` de un objeto `DB2Transaction`.

Retrotracción o confirmación de una transacción en C#:

```
// se supone la utilización del objeto DB2Transaction conn
trans.Rollback();
...
trans.Commit();
```

Retrotracción o confirmación de una transacción en C#:

```
' se supone la utilización del objeto DB2Transaction conn
trans.Rollback();
...
trans.Commit()
```

Lectura de conjuntos de resultados de una aplicación utilizando DB2 .NET Data Provider

Cuando se utiliza DB2 .NET Data Provider, la lectura de los conjuntos de resultados se realiza mediante un objeto `DB2DataReader`. Se utiliza el método `Read()` de `DB2DataReader` para avanzar hacia la fila siguiente del conjunto de

resultados. Para extraer los datos de las columnas del resultado se utilizan los métodos GetString(), GetInt32(), GetDecimal(), y otros métodos existentes para todos los tipos de datos disponibles. El método Close() de DB2DataReader se utiliza para cerrar el objeto DB2DataReader, lo cual debe hacerse siempre al terminar de leer el resultado.

Lectura de un conjunto de resultados en C#:

```
// se supone la utilización de un lector DB2DataReader
Int16 deptnum = 0;
String location="";

// Visualizar los resultados de la consulta
while(reader.Read())
{
  deptnum = reader.GetInt16(0);
  location = reader.GetString(1);
  Console.WriteLine("    " + deptnum + " " + location);
}
reader.Close();
```

Lectura de un conjunto de resultados en Visual Basic .NET:

```
' se supone la utilización de un lector DB2DataReader
Dim deptnum As Int16 = 0
Dim location As String ""

' Visualizar los resultados de la consulta
Do While (reader.Read())
  deptnum = reader.GetInt16(0)
  location = reader.GetString(1)
  Console.WriteLine("    " & deptnum & " " & location)
Loop
reader.Close();
```

Invocación de procedimientos almacenados desde una aplicación utilizando DB2 .NET Data Provider

Cuando utiliza DB2 .NET Data Provider, puede invocar procedimientos almacenados utilizando un objeto DB2Command. El valor por omisión de la propiedad CommandType es CommandType.Text. Este valor es el apropiado para sentencias de SQL y también se puede utilizar para invocar procedimientos almacenados. Pero la invocación de procedimientos almacenados es más fácil si define el valor de CommandType como CommandType.StoredProcedure. En este caso, solo es necesario que especifique el nombre del procedimiento almacenado y los parámetros.

Los ejemplos siguientes muestran cómo invocar un procedimiento almacenado llamado INOUT_PARAM, con la propiedad CommandType establecida en CommandType.StoredProcedure o CommandType.Text.

Invocación de un procedimiento almacenado utilizando CommandType.StoredProcedure como valor de la propiedad CommandType de DB2Command en C#:

```
// se supone la existencia de una conexión DB2Connection
DB2Transaction trans = conn.BeginTransaction();
DB2Command cmd = conn.CreateCommand();
String procName = "INOUT_PARAM";
cmd.Transaction = trans;
cmd.CommandType = CommandType.StoredProcedure;
cmd.CommandText = procName;
```

```
// Registro de los parámetros de entrada-salida y parámetros de
salida para DB2Command
...

// Invocación del procedimiento almacenado
Console.WriteLine("  Call stored procedure named " + procName);
cmd.ExecuteNonQuery();
```

Invocación de un procedimiento almacenado utilizando CommandType.Text como valor de la propiedad CommandType de DB2Command en C#:

```
// se supone la existencia de una conexión DB2Connection
DB2Transaction trans = conn.BeginTransaction();
DB2Command cmd = conn.CreateCommand();
String procName = "INOUT_PARAM";
String procCall = "CALL INOUT_PARAM (?, ?, ?)";
cmd.Transaction = trans;
cmd.CommandType = CommandType.Text;
cmd.CommandText = procCall;

// Registro de los parámetros de entrada-salida y parámetros de
salida para DB2Command
...

// Invocación del procedimiento almacenado
Console.WriteLine("  Call stored procedure named " + procName);
cmd.ExecuteNonQuery();
```

Invocación de un procedimiento almacenado utilizando CommandType.StoredProcedure como valor de la propiedad CommandType de DB2Command en Visual Basic .NET:

```
' se supone la utilización de un lector DB2DataReader
Dim trans As DB2Transaction = conn.BeginTransaction()
Dim cmd As DB2Command = conn.CreateCommand()
Dim procName As String = "INOUT_PARAM"
cmd.Transaction = trans
cmd.CommandType = CommandType.StoredProcedure
cmd.CommandText = procName

' Registro de los parámetros de entrada-salida y parámetros de
salida para DB2Command
...

' Invocación del procedimiento almacenado
Console.WriteLine("  Call stored procedure named " & procName)
cmd.ExecuteNonQuery()
```

Invocación de un procedimiento almacenado utilizando CommandType.Text como valor de la propiedad CommandType de DB2Command en Visual Basic .NET:

```
' se supone la utilización de un lector DB2DataReader
Dim trans As DB2Transaction = conn.BeginTransaction()
Dim cmd As DB2Command = conn.CreateCommand()
Dim procName As String = "INOUT_PARAM"
Dim procCall As String = "CALL INOUT_PARAM (?, ?, ?)"
cmd.Transaction = trans
cmd.CommandType = CommandType.Text
cmd.CommandText = procCall

' Registro de los parámetros de entrada-salida y parámetros de salida
para DB2Command
...

' Invocación del procedimiento almacenado
Console.WriteLine("  Call stored procedure named " & procName)
cmd.ExecuteNonQuery()
```

Tipos de datos SQL soportados para DB2 .NET Data Provider

La tabla siguiente muestra las correlaciones existentes entre los tipos de datos `DB2Type` de DB2 .NET Data Provider, el tipo de datos DB2 y el correspondiente tipo de datos de .NET Framework.

Tabla 17. Correlaciones entre tipos de datos DB2 y tipos de datos .NET

Enumerador de DB2Type	Tipo de datos de DB2	Tipo de datos de .NET
SmallInt	SMALLINT	Int16
Integer	INTEGER	Int32
BigInt	BIGINT	Int64
Real	REAL	Single
Double	DOUBLE PRECISION	Double
Float	FLOAT	Double
Decimal	DECIMAL	Decimal
Numeric	DECIMAL	Decimal
Date	DATE	DateTime
Time	TIME	TimeSpan
Timestamp	TIMESTAMP	DateTime
Char	CHAR	String
VarChar	VARCHAR	String
LongVarChar(1)	LONG VARCHAR	String
Binary	CHAR FOR BIT DATA	Byte[]
VarBinary	VARCHAR FOR BIT DATA	Byte[]
LongVarBinary(1)	LONG VARCHAR FOR BIT DATA	Byte[]
Graphic	GRAPHIC	String
VarGraphic	VARGRAPHIC	String
LongVarGraphic(1)	LONG GRAPHIC	String
Clob	CLOB	String
Blob	BLOB	Byte[]
DbClob	DBCLOB(N)	String

Notas:

1. Estos tipos de datos no están soportados en las rutinas CLR (common language runtime) de DB2 .NET. Solo están soportados en aplicaciones cliente.

Nota: La estructura `dbinfo` se pasa como parámetro a funciones y procedimientos de CLR. La memoria de trabajo y tipo de llamada de las UDF de CLR también se pasan como parámetros a las rutinas CLR. Para obtener información sobre los tipos de datos apropiados de CLR para esos parámetros, consulte el tema asociado:

- Parámetros de rutinas CLR

Conceptos relacionados:

- "Estilos de parámetros para rutinas externas" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*

- "Rutinas CLR (Common Language Runtime)" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*
- "Parámetros en las rutinas CLR" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*

Tareas relacionadas:

- "Pase de parámetros de tipo estructurado a rutinas externas" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*
- "Creación de rutinas CLR" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*
- "Ejemplos de funciones definidas por el usuario CLR en C#" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*
- "Ejemplos de procedimientos CLR en C#" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*

Ejemplos relacionados:

- "SpCreate.db2 -- Creates the external procedures implemented in spserver.cs"
- "SpServer.cs -- C# external code implementation of procedures created in spcat.db2"
- "SpCreate.db2 -- Creates the external procedures implemented in spserver.vb"
- "SpServer.vb -- VB.NET implementation of procedures created in SpCat.db2"

Capítulo 11. IBM OLE DB Provider para DB2

Objetivo de IBM OLE DB Provider para DB2 . . . 241
Tipos de aplicaciones soportados por IBM OLE DB Provider para DB2. 242
Servicios OLE DB 242
Modelo de hebras soportado por IBM OLE DB Provider 242
Manipulación de objetos grandes con IBM OLE DB Provider 243
Conjuntos de filas de esquema soportados por IBM OLE DB Provider 243
Habilitación automática de servicios OLE DB por parte de IBM OLE DB Provider 245
Servicios de datos 245
Modalidades de cursor soportadas en IBM OLE DB Provider 245
Correlaciones de tipos de datos entre DB2 y OLE DB 245
Conversión de datos para establecer datos de tipos OLE DB en tipos DB2. 246
Conversión de datos para establecer datos de tipos DB2 en tipos OLE DB. 248
Restricciones de IBM OLE DB Provider. 250
Soporte de IBM OLE DB Provider de interfaces y componentes de OLE DB 250
Soporte de IBM OLE DB Provider para propiedades de OLE DB. 253
Conexiones a fuentes de datos mediante IBM OLE DB Provider 255
Aplicaciones ADO. 256
Palabras clave de series de conexión de ADO 256
Conexiones con fuentes de datos con aplicaciones ADO Visual Basic. 257
Cursores desplazables actualizables en aplicaciones ADO 257
Limitaciones de las aplicaciones ADO 257
Soporte de IBM OLE DB Provider de propiedades y métodos ADO 258
Aplicaciones C y C++ 261
Compilación y enlace de aplicaciones C/C++ e IBM OLE DB Provider 261
Conexiones con fuentes de datos en aplicaciones C/C++ mediante IBM OLE DB Provider . . . 262
Transacciones distribuidas MTS y COM+ 262
Soporte de transacciones distribuidas MTS y COM+ e IBM OLE DB Provider 262
Habilitación del soporte de MTS en DB2 Universal Database para aplicaciones C/C++. . 263

Objetivo de IBM OLE DB Provider para DB2

Microsoft® OLE DB es un conjunto de interfaces OLE/COM que proporciona a las aplicaciones un acceso uniforme a datos almacenados en distintas fuentes de información. La arquitectura OLE DB define consumidores de OLE DB y proveedores de OLE DB. Un consumidor de OLE DB es cualquier sistema o aplicación que utiliza interfaces OLE DB; un proveedor de OLE DB es un componente que expone las interfaces OLE DB.

IBM® OLE DB Provider para DB2® permite a DB2 actuar como gestor de recursos para el proveedor de OLE DB. Este soporte ofrece a las aplicaciones basadas en OLE DB la posibilidad de extraer o consultar datos de DB2 mediante la interfaz OLE. IBM OLE DB Provider para DB2, cuyo nombre de proveedor es IBMDADB2, permite a los consumidores de OLE DB acceder a datos en un servidor DB2 Universal Database™. Si DB2 Connect™ está instalado, estos consumidores de OLE DB también pueden acceder a datos contenidos en sistemas principales DBMS, tales como DB2 para MVS™, DB2 para VM/VSE o SQL/400.

IBM OLE DB Provider para DB2 ofrece las siguientes funciones:

- Nivel de soporte 0 de la especificación de proveedor de OLE DB, incluidas algunas interfaces adicionales de nivel 1.
- Una implantación del proveedor de hebras libres, que permite a la aplicación crear componentes en una hebra y utilizar dichos componentes en cualquier otra hebra.
- Un Servicio de búsqueda de errores que devuelve mensajes de error de DB2.

Tenga en cuenta que IBM OLE DB Provider reside en el cliente y es distinto de las funciones de tabla de OLE DB, que también reciben soporte de DB2 UDB.

Las siguientes secciones de este documento describen la implantación específica de IBM OLE DB Provider para DB2. Para obtener más información sobre la especificación OLE DB 2.0 de Microsoft, consulte el manual ″Microsoft OLE DB 2.0 Programmer's Reference and Data Access SDK″, publicado por Microsoft Press.

Cumplimiento de versiones:

IBM OLE DB Provider para DB2 se ajusta a la Versión 2.7 de la especificación OLE DB de Microsoft.

Requisitos del sistema:

Consulte la carta de anuncio correspondiente a IBM OLE DB Provider para DB2 Servers para ver los sistemas operativos Windows® soportados.

Para instalar IBM OLE DB Provider para DB2, primero tiene que ejecutar uno de los sistemas operativos soportados mencionados anteriormente. También tiene que instalar DB2 Application Development Client, así como Microsoft Data Access Components (MDAC) Versión 2.7, o superior, que estaba disponible en el momento de escribir este documento en el siguiente sitio: `http://www.microsoft.com/data`.

Información relacionada:

- "Soporte de IBM OLE DB Provider de interfaces y componentes de OLE DB" en la página 250

Tipos de aplicaciones soportados por IBM OLE DB Provider para DB2

Con IBM® OLE DB Provider para DB2®, puede crear los siguientes tipos de aplicaciones:

- Aplicaciones ADO, que incluyen:
  - Aplicaciones Microsoft® Visual Studio C++
  - Aplicaciones Microsoft Visual Basic
- Aplicaciones ADO.NET que hacen uso de OLE DB .NET Data Provider
- Aplicaciones C/C++ que acceden a IBMDADB2 directamente mediante las interfaces OLE DB, incluidas aplicaciones ATL cuyos Objetos de consumidor de acceso a datos se han generado mediante ATL COM AppWizard.

Servicios OLE DB

Las secciones siguientes describen los servicios OLE DB.

Modelo de hebras soportado por IBM OLE DB Provider

IBM® OLE DB Provider para DB2® da soporte al modelo de hebras libres, que permite a las aplicaciones crear componentes en una hebra y utilizar dichos componentes en cualquier otra hebra.

Manipulación de objetos grandes con IBM OLE DB Provider

Para obtener y establecer datos como objetos de almacenamiento (DBTYPE_IUNKNOWN) con IBMDADB2, utilice la interfaz ISequentialStream del siguiente modo:

- Para vincular un objeto de almacenamiento a un parámetro, DBOBJECT en la estructura DBBINDING sólo puede contener el valor STGM_READ para el campo dwFlag. IBMDADB2 ejecutará el método Read de la interfaz ISequentialStream del objeto vinculado.
- Para obtener datos de un objeto de almacenamiento, la aplicación debe ejecutar un método Read en la interfaz ISequentialStream del objeto de almacenamiento.
- Cuando se obtienen datos, el valor de la parte de longitud es la longitud de los datos reales, no la longitud del puntero IUnknown.

Conjuntos de filas de esquema soportados por IBM OLE DB Provider

La tabla siguiente muestra los conjuntos de filas de esquema soportados por `IDBSchemaRowset`. Tenga que cuenta que las columnas no soportadas se establecerán en nulo en los conjuntos de filas.

Tabla 18. Conjuntos de filas soportados por IBM OLE DB Provider para DB2

GUID soportados	Restricciones soportadas	Columnas soportadas	Notas
DBSCHEMA _COLUMN_PRIVILEGES	COLUMN_NAME TABLE_NAME TABLE_SCHEMA	COLUMN_NAME GRANTEE GRANTOR IS_GRANTABLE PRIVILEGE_TYPE TABLE_NAME TABLE_SCHEMA	
DB_SCHEMA_COLUMNS	COLUMN_NAME TABLE_NAME TABLE_SCHEMA	CHARACTER_MAXIMUM_LENGTH CHARACTER_OCTET_LENGTH COLUMN_DEFAULT COLUMN_FLAGS COLUMN_HASDEFAULT COLUMN_NAME DATA_TYPE DESCRIPTION IS_NULLABLE NUMERIC_PRECISION NUMERIC_SCALE ORDINAL_POSITION TABLE_NAME TABLE_SCHEMA	
DBSCHEMA_FOREIGN_KEYS	FK_TABLE_NAME FK_TABLE_SCHEMA PK_TABLE_NAME PK_TABLE_SCHEMA	DEFERRABILITY DELETE_RULE FK_COLUMN_NAME FK_NAME FK_TABLE_NAME FK_TABLE_SCHEMA ORDINAL PK_COLUMN_NAME PK_NAME PK_TABLE_NAME PK_TABLE_SCHEMA UPDATE_RULE	Se debe especificar al menos una de las siguientes restricciones: PK_TABLE_NAME o FK_TABLE_NAME No se permiten caracteres comodín "%".

Tabla 18. Conjuntos de filas soportados por IBM OLE DB Provider para DB2 (continuación)

GUID soportados	Restricciones soportadas	Columnas soportadas	Notas
DBSCHEMA_INDEXES	TABLE_NAME TABLE_SCHEMA	CARDINALITY CLUSTERED COLLATION COLUMN_NAME INDEX_NAME INDEX_SCHEMA ORDINAL_POSITION PAGES TABLE_NAME TABLE_SCHEMA TYPE UNIQUE	No se permite ningún orden de clasificación. El orden de clasificación, si se especifica, se pasará por alto.
DBSCHEMA_PRIMARY_KEYS	TABLE_NAME TABLE_SCHEMA	COLUMN_NAME ORDINAL PK_NAME TABLE_NAME TABLE_SCHEMA	Se debe especificar al menos la siguiente restricción: TABLE_NAME No se permiten caracteres comodín "%".
DBSCHEMA _PROCEDURE_PARAMETERS	PARAMETER_NAME PROCEDURE_NAME PROCEDURE_SCHEMA	CHARACTER_MAXIMUM_LENGTH CHARACTER_OCTET_LENGTH DATA_TYPE DESCRIPTION IS_NULLABLE NUMERIC_PRECISION NUMERIC_SCALE ORDINAL_POSITION PARAMETER_DEFAULT PARAMETER_HASDEFAULT PARAMETER_NAME PARAMETER_TYPE PROCEDURE_NAME PROCEDURE_SCHEMA TYPE_NAME	
DBSCHEMA_PROCEDURES	PROCEDURE_NAME PROCEDURE_SCHEMA	DESCRIPTION PROCEDURE_NAME PROCEDURE_SCHEMA PROCEDURE_TYPE	
DBSCHEMA_PROVIDER_TYPES	DATA_TYPE BEST_MATCH	AUTO_UNIQUE_VALUE BEST_MATCH CASE_SENSITIVE CREATE_PARAMS COLUMN_SIZE DATA_TYPE FIXED_PREC_SCALE IS_FIXEDLENGTH IS_LONG IS_NULLABLE LITERAL_PREFIX LITERAL_SUFFIX LOCAL_TYPE_NAME MINIMUM_SCALE MAXIMUM_SCALE SEARCHABLE TYPE_NAME UNSIGNED_ATTRIBUTE	
DBSCHEMA_STATISTICS	TABLE_NAME TABLE_SCHEMA	CARDINALITY TABLE_NAME TABLE_SCHEMA	No se permite ningún orden de clasificación. El orden de clasificación, si se especifica, se pasará por alto.
DBSCHEMA _TABLE_PRIVILEGES	TABLE_NAME TABLE_SCHEMA	GRANTEE GRANTOR IS_GRANTABLE PRIVILEGE_TYPE TABLE_NAME TABLE_SCHEMA	
DBSCHEMA_TABLES	TABLE_NAME TABLE_SCHEMA TABLE_TYPE	DESCRIPTION TABLE_NAME TABLE_SCHEMA TABLE_TYPE	

Habilitación automática de servicios OLE DB por parte de IBM OLE DB Provider

Por omisión, IBM® OLE DB Provider para DB2® habilita de forma automática todos los servicios OLE DB, añadiendo una entrada de registro OLEDB_SERVICES bajo el ID de clase (CLSID) del proveedor con el valor 0xFFFFFFFF para DWORD. El significado de este valor es el siguiente:

Tabla 19. Servicios OLE DB

Servicios habilitados	Valor de DWORD
Todos los servicios (valor por omisión)	0xFFFFFFFF
Todos excepto agrupación y AutoEnlistment	0xFFFFFFFC
Todos excepto cursor del cliente	0xFFFFFFFB
Todos excepto agrupación, alistamiento y cursor	0xFFFFFFF8
Ningún servicio	0x000000000

Servicios de datos

Las secciones siguientes describen consideraciones sobre los servicios de datos.

Modalidades de cursor soportadas en IBM OLE DB Provider

IBM® OLE DB Provider para DB2® da soporte nativo a cursores de sólo lectura, de sólo avance, y a cursores desplazables y actualizables.

Correlaciones de tipos de datos entre DB2 y OLE DB

IBM OLE DB Provider da soporte a las correlaciones de tipos de datos entre tipos de datos DB2 y tipos de datos OLE DB. La tabla siguiente proporciona una lista completa de correlaciones soportadas y nombres disponibles para indicar los tipos de datos de columnas y parámetros.

Tabla 20. Correlaciones de tipos de datos entre tipos de datos DB2 y tipos de datos OLE DB

Tipos de datos DB2	Indicadores de tipos de datos OLE DB	Nombres de tipos estándar de OLE DB	Nombres específicos de DB2
SMALLINT	DBTYPE_I2	"DBTYPE_I2"	"SMALLINT"
INTEGER	DBTYPE_I4	"DBTYPE_I4"	"INTEGER" o "INT"
BIGINT	DBTYPE_I8	"DBTYPE_I8"	"BIGINT"
REAL	DBTYPE_R4	"DBTYPE_R4"	"REAL"
FLOAT	DBTYPE_R8	"DBTYPE_R8"	"FLOAT"
DOUBLE	DBTYPE_R8	"DBTYPE_R8"	"DOUBLE" o "DOUBLE PRECISION"
DECIMAL	DBTYPE_NUMERIC	"DBTYPE_NUMERIC"	"DEC" o "DECIMAL"
NUMERIC	DBTYPE_NUMERIC	"DBTYPE_NUMERIC"	"NUM" o "NUMERIC"
DATE	DBTYPE_DBDATE	"DBTYPE_DBDATE"	"DATE"
TIME	DBTYPE_DBTIME	"DBTYPE_DBTIME"	"TIME"
TIMESTAMP	DBTYPE_DBTIMESTAMP	"DBTYPE_DBTIMESTAMP"	"TIMESTAMP"
CHAR	DBTYPE_STR	"DBTYPE_CHAR"	"CHAR" o "CHARACTER"
VARCHAR	DBTYPE_STR	"DBTYPE_VARCHAR"	"VARCHAR"

Tabla 20. Correlaciones de tipos de datos entre tipos de datos DB2 y tipos de datos OLE DB (continuación)

Tipos de datos DB2	Indicadores de tipos de datos OLE DB	Nombres de tipos estándar de OLE DB	Nombres específicos de DB2
LONG VARCHAR	DBTYPE_STR	"DBTYPE_LONGVARCHAR"	"LONG VARCHAR"
CLOB	DBTYPE_STR y DBCOLUMNFLAGS_ISLONG o DBPARAMFLAGS_ISLONG	"DBTYPE_CHAR" "DBTYPE_VARCHAR" "DBTYPE_LONGVARCHAR" y DBCOLUMNFLAGS_ISLONG o DBPARAMFLAGS_ISLONG	"CLOB"
GRAPHIC	DBTYPE_WSTR	"DBTYPE_WCHAR"	"GRAPHIC"
VARGRAPHIC	DBTYPE_WSTR	"DBTYPE_WVARCHAR"	"VARGRAPHIC"
LONG VARGRAPHIC	DBTYPE_WSTR	"DBTYPE_WLONGVARCHAR"	"LONG VARGRAPHIC"
DBCLOB	DBTYPE_WSTR y DBCOLUMNFLAGS_ISLONG o DBPARAMFLAGS_ISLONG	"DBTYPE_WCHAR" "DBTYPE_WVARCHAR" "DBTYPE_WLONGVARCHAR" y DBCOLUMNFLAGS_ISLONG o DBPARAMFLAGS_ISLONG	"DBCLOB"
CHAR(n) FOR BIT DATA	DBTYPE_BYTES	"DBTYPE_BINARY"	
VARCHAR(n) FOR BIT DATA	DBTYPE_BYTES	"DBTYPE_VARBINARY"	
LONG VARCHAR FOR BIT DATA	DBTYPE_BYTES	"DBTYPE_LONGVARBINARY"	
BLOB	DBTYPE_BYTES y DBCOLUMNFLAGS_ISLONG o DBPARAMFLAGS_ISLONG	"DBTYPE_BINARY" "DBTYPE_VARBINARY" "DBTYPE_LONGVARBINARY" y DBCOLUMNFLAGS_ISLONG o DBPARAMFLAGS_ISLONG	"BLOB"
DATA LINK	DBTYPE_STR	"DBTYPE_CHAR"	"DATA LINK"

Conversión de datos para establecer datos de tipos OLE DB en tipos DB2

IBM OLE DB Provider da soporte a las conversiones de datos para establecer datos de tipos OLE DB en tipos DB2. Tenga en cuenta que es posible que se trunquen datos en algunos casos, en función de los tipos y del valor de los datos.

Tabla 21. Conversión de datos de tipos OLE DB a tipos DB2

	Tipos de datos DB2																						
																		For Bit Data					
Indicador de tipo OLE DB	**SMALLINT**	**INTEGER**	**BIGINT**	**REAL**	**FLOAT DOUBLE**	**DECIMAL NUMERIC**	**DATE**	**TIME**	**TIMESTAMP**	**CHAR**	**VARCHAR**	**LONG VARCHAR**	**CLOB**	**GRAPHIC**	**VARGRAPHIC**	**LONG VARGRAPHIC**	**DBCLOB**	**CHAR**	**VARCHAR**	**LONG VARCHAR**	**BLOB**	**DATALINK**
DBTYPE_EMPTY																							
DBTYPE_NULL																							
DBTYPE_RESERVED																							
DBTYPE_I1	X	X	X	X	X	X				X	X												
DBTYPE_I2	X	X	X	X	X	X				X	X												
DBTYPE_I4	X	X	X	X	X	X				X	X												
DBTYPE_I8	X	X	X	X	X	X				X	X												
DBTYPE_UI1	X	X	X	X	X	X				X	X												
DBTYPE_UI2	X	X	X	X	X	X				X	X												
DBTYPE_UI4	X	X	X	X	X	X				X	X												
DBTYPE_UI8	X	X	X	X	X	X				X	X												
DBTYPE_R4	X	X	X	X	X	X				X	X												
DBTYPE_R8	X	X	X	X	X	X				X	X												
DBTYPE_CY																							
DBTYPE_DECIMAL	X	X	X	X	X	X				X	X												
DBTYPE_NUMERIC	X	X	X	X	X	X				X	X												
DBTYPE_DATE																							
DBTYPE_BOOL	X	X	X	X	X	X				X	X												
DBTYPE_BYTES			X			X				X	X	X				X		X	X	X			
DBTYPE_BSTR – por determinar																							
DBTYPE_STR	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X	
DBTYPE_WSTR														X	X	X							
DBTYPE_VARIANT – por determinar																							
DBTYPE_IDISPATCH																							
DBTYPE_IUNKNOWN										X	X	X	X	X	X	X	X	X	X	X			
DBTYPE_GUID																							
DBTYPE_ERROR																							
DBTYPE_BYREF																							
DBTYPE_ARRAY																							
DBTYPE_VECTOR																							
DBTYPE_UDT																							
DBTYPE_DBDATE							X		X	X	X												
DBTYPE_DBTIME								X	X	X	X												

Tabla 21. Conversión de datos de tipos OLE DB a tipos DB2 (continuación)

Indicador de tipo OLE DB	Tipos de datos DB2																					
	SMALLINT	INTEGER	BIGINT	REAL	FLOAT DOUBLE	DECIMAL NUMERIC	DATE	TIME	TIMESTAMP	CHAR	VARCHAR	LONG VARCHAR	CLOB	GRAPHIC	VARGRAPHIC	LONG VARGRAPHIC	DBCLOB	For Bit Data			BLOB	DATALINK
																		CHAR	VARCHAR	LONG VARCHAR		
DBTYPE_DBTIMESTAMP							X	X	X	X	X											
DBTYPE_FILETIME																						
DBTYPE_PROP_VARIANT																						
DBTYPE_HCHAPTER																						
DBTYPE_VARNUMERIC																						

Información relacionada:

- "Conversión de datos para establecer datos de tipos DB2 en tipos OLE DB" en la página 248

Conversión de datos para establecer datos de tipos DB2 en tipos OLE DB

Para obtener datos, IBM OLE DB Provider permite conversiones de datos de tipos DB2 a tipos OLE DB. Tenga en cuenta que es posible que se trunquen datos en algunos casos, en función de los tipos y del valor de los datos.

Tabla 22. Conversiones de datos de tipos DB2 a tipos OLE DB

Indicador de tipo OLE DB	Tipos de datos DB2																					
	SMALLINT	INTEGER	BIGINT	REAL	FLOAT DOUBLE	DECIMAL NUMERIC	DATE	TIME	TIMESTAMP	CHAR	VARCHAR	LONG VARCHAR	CLOB	GRAPHIC	VARGRAPHIC	LONG VARGRAPHIC	DBCLOB	For Bit Data			BLOB	DATALINK
																		CHAR	VARCHAR	LONG VARCHAR		
DBTYPE_EMPTY																						
DBTYPE_NULL																						
DBTYPE_RESERVED																						
DBTYPE_I1	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_I2	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X

Tabla 22. Conversiones de datos de tipos DB2 a tipos OLE DB (continuación)

	Tipos de datos DB2																						
																			For Bit Data				
Indicador de tipo OLE DB	**SMALLINT**	**INTEGER**	**BIGINT**	**REAL**	**FLOAT DOUBLE**	**DECIMAL NUMERIC**	**DATE**	**TIME**	**TIMESTAMP**	**CHAR**	**VARCHAR**	**LONG VARCHAR**	**CLOB**	**GRAPHIC**	**VARGRAPHIC**	**LONG VARGRAPHIC**	**DBCLOB**	**CHAR**	**VARCHAR**	**LONG VARCHAR**	**BLOB**	**DATALINK**
DBTYPE_I4	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_I8	X	X	X	X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_UI1	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_UI2	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_UI4	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_UI8	X	X	X	X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_R4	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_R8	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_CY	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_DECIMAL	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_NUMERIC	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_DATE	X	X		X	X		X	X	X	X	X	X		X	X	X						X
DBTYPE_BOOL	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_BYTES	X	X		X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_BSTR	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_STR	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_WSTR	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_VARIANT	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_IDISPATCH																						
DBTYPE_IUNKNOWN	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
DBTYPE_GUID										X	X	X		X	X	X		X	X	X		X
DBTYPE_ERROR																						
DBTYPE_BYREF																						
DBTYPE_ARRAY																						
DBTYPE_VECTOR																						
DBTYPE_UDT																						
DBTYPE_DBDATE							X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_DBTIME							X	X	X	X	X	X		X	X	X						X
DBTYPE_DBTIMESTAMP							X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_FILETIME			X				X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_PROP_VARIANT	X	X	X	X	X					X	X	X		X	X	X		X	X	X		X
DBTYPE_HCHAPTER																						
DBTYPE_VARNUMERIC																						

Tabla 22. Conversiones de datos de tipos DB2 a tipos OLE DB (continuación)

Indicador de tipo OLE DB	Tipos de datos DB2																				
	SMALLINT	INTEGER	BIGINT	REAL	FLOAT DOUBLE	DECIMAL NUMERIC	DATE	TIME	TIMESTAMP	CHAR	VARCHAR	LONG VARCHAR	CLOB	GRAPHIC	VARGRAPHIC	LONG VARGRAPHIC	DBCLOB	For Bit Data			BLOB	DATALINK
																		CHAR	VARCHAR	LONG VARCHAR		

Nota: Cuando la aplicación realiza una operación ISequentialStream::Read para obtener los datos del objeto de almacenamiento, el formato de los datos devueltos depende del tipo de datos de la columna:

- Para tipos de datos binarios y que no sean de tipo carácter, los datos de la columna se exponen como una secuencia de bytes que representan dichos valores en el sistema operativo.
- Para tipos de datos de carácter, primero los datos se convierten a DBTYPE_STR.
- Para DBCLOB, primero los datos se convierten a DBTYPE_WCHAR.

Información relacionada:

- "Conversión de datos para establecer datos de tipos OLE DB en tipos DB2" en la página 246

Restricciones de IBM OLE DB Provider

A continuación se describen las restricciones de IBM® OLE DB Provider:

- IBMDADB2 da soporte al ámbito de transacciones de confirmación automática y controladas por el usuario con la interfaz `ITransactionLocal`. El ámbito de transacciones de confirmación automática es el ámbito por omisión. Las transacciones anidadas no reciben soporte.
- `RestartPosition` no recibe soporte si el texto del mandato contiene parámetros.
- IBMDADB2 no pone entre comillas nombres de tablas que se pasan a través de parámetros de DBID, que son los parámetros que utiliza la interfaz `IOpenRowset`. En su lugar, el consumidor de OLE DB debe añadir comillas a los nombres de tablas cuando sea necesario.

Soporte de IBM OLE DB Provider de interfaces y componentes de OLE DB

La tabla siguiente contiene las interfaces y componentes de OLE DB que reciben soporte de IBM OLE DB Provider y de Microsoft OLE DB Provider para ODBC.

Tabla 23. Comparación de interfaces y componentes de OLE DB soportados por IBM OLE DB Provider para DB2 y Microsoft OLE DB Provider para ODBC

	Interfaz	DB2	ODBC Provider
BLOB			
	ISequentialStream	Sí	Sí
Mandato			

Tabla 23. Comparación de interfaces y componentes de OLE DB soportados por IBM OLE DB Provider para DB2 y Microsoft OLE DB Provider para ODBC (continuación)

	Interfaz	DB2	ODBC Provider
	IAccessor	Sí	Sí
	ICommand	Sí	Sí
	ICommandPersist	No	No
	ICommandPrepare	Sí	Sí
	ICommandProperties	Sí	Sí
	ICommandText	Sí	Sí
	ICommandWithParameters	Sí	Sí
	IColumnsInfo	Sí	Sí
	IColumnsRowset	Sí	Sí
	IConvertType	Sí	Sí
	ISupportErrorInfo	Sí	Sí
Fuente de datos			
	IConnectionPoint	No	Sí
	IDBAsynchNotify (consumer)	No	No
	IDBAsynchStatus	No	No
	IDBConnectionPointContainer	No	Sí
	IDBCreateSession	Sí	Sí
	IDBDataSourceAdmin	No	No
	IDBInfo	Sí	Sí
	IDBInitialize	Sí	Sí
	IDBProperties	Sí	Sí
	IPersist	Sí	No
	IPersistFile	Sí	Sí
	ISupportErrorInfo	Sí	Sí
Enumerador			
	IDBInitialize	Sí	Sí
	IDBProperties	Sí	Sí
	IParseDisplayName	Sí	No
	ISourcesRowset	Sí	Sí
	ISupportErrorInfo	Sí	Sí
Servicio de búsqueda de errores			
	IErrorLookUp	Sí	Sí
Objeto Error			
	IErrorInfo	Sí	No
	IErrorRecords	Sí	No
	ISQLErrorInfo (custom)	Sí	No
Varios resultados			
	IMultipleResults	Sí	Sí
	ISupportErrorInfo	Sí	Sí

Tabla 23. Comparación de interfaces y componentes de OLE DB soportados por IBM OLE DB Provider para DB2 y Microsoft OLE DB Provider para ODBC (continuación)

	Interfaz	DB2	ODBC Provider
Conjunto de filas			
	IAccessor	Sí	Sí
	IColumnsRowset	Sí	Sí
	IColumnsInfo	Sí	Sí
	IConvertType	Sí	Sí
	IChapteredRowset	No	No
	IConnectionPointContainer	Sí	Sí
	IDBAsynchStatus	No	No
	IParentRowset	No	No
	IRowset	Sí	Sí
	IRowsetChange	Sí	Sí
	IRowsetChapterMember	No	No
	IRowsetFind	No	No
	IRowsetIdentity	Sí	Sí
	IRowsetIndex	No	No
	IRowsetInfo	Sí	Sí
	IRowsetLocate	Sí	Sí
	IRowsetNotify (consumer)	Sí	No
	IRowsetRefresh	Componente Servicio de cursor	Sí
	IRowsetResynch	Componente Servicio de cursor	Sí
	IRowsetScroll	Sí[1]	Sí
	IRowsetUpdate	Componente Servicio de cursor	Sí
	IRowsetView	No	No
	ISupportErrorInfo	Sí	Sí
Notas: 1. Los valores a devolver son aproximaciones. Las filas suprimidas no se pasarán por alto.			
Sesión			
	IAlterIndex	No	No
	IAlterTable	No	No
	IDBCreateCommand	Sí	Sí
	IDBSchemaRowset	Sí	Sí
	IGetDataSource	Sí	Sí
	IIndexDefinition	No	No
	IOpenRowset	Sí	Sí
	ISessionProperties	Sí	Sí
	ISupportErrorInfo	Sí	Sí
	ITableDefinition	No	No
	ITableDefinitionWithConstraints	No	No
	ITransaction	Sí	Sí

Tabla 23. Comparación de interfaces y componentes de OLE DB soportados por IBM OLE DB Provider para DB2 y Microsoft OLE DB Provider para ODBC (continuación)

	Interfaz	DB2	ODBC Provider
	ITransactionJoin	Sí	Sí
	ITransactionLocal	Sí	Sí
	ITransactionObject	No	No
	ITransactionOptions	No	Sí
Objetos Vista			
	IViewChapter	No	No
	IViewFilter	No	No
	IViewRowset	No	No
	IViewSort	No	No

Soporte de IBM OLE DB Provider para propiedades de OLE DB

La tabla siguiente muestra las propiedades de OLE DB que reciben soporte de IBM OLE DB Provider:

Tabla 24. Propiedades soportadas por IBM OLE DB Provider para DB2

Grupo de propiedades	Conjunto de propiedades	Propiedades	Valor por omisión	R/W
Fuente de datos	DBPROPSET_DATASOURCE	DBPROP_MULTIPLECONNECTIONS	VARIANT_FALSE	R
		DBPROP_RESETDATASOURCE	DBPROPVAL_RD_RESETALL	R/W
Información de fuente de datos	DBPROPSET _DATASOURCEINFO	DBPROP_ACTIVESESSIONS	0	R
		DBPROP_ASYNCTXNABORT	VARIANT_FALSE	R
		DBPROP_ASYNCTXNCOMMIT	VARIANT_FALSE	R
		DBPROP_BYREFACCESSORS	VARIANT_FALSE	R
		DBPROP_COLUMNDEFINITION	DBPROPVAL_CD_NOTNULL	R
		DBPROP_CONCATNULLBEHAVIOR	DBPROPVAL_CB_NULL	R
		DBPROP_CONNECTIONSTATUS	DBPROPVAL_CS_INITIALIZED	R
		DBPROP_DATASOURCENAME	N/D	R
		DBPROP_DATASOURCEREADONLY	VARIANT_FALSE	R
		DBPROP_DBMSNAME	N/D	R
		DBPROP_DBMSVER	N/D	R
		DBPROP_DSOTHREADMODEL	DBPROPVAL_RT_FREETHREAD	R
		DBPROP_GROUPBY	DBPROPVAL_GB_CONTAINS_SELECT	R
		DBPROP_IDENTIFIERCASE	DBPROPVAL_IC_UPPER	R
		DBPROP_MAXINDEXSIZE	0	R
		DBPROP_MAXROWSIZE	0	R
		DBPROP_MAXROWSIZEINCLUDESBLOB	VARIANT_TRUE	R
		DBPROP_MAXTABLEINSELECT	0	R
		DBPROP_MULTIPLEPARAMSETS	VARIANT_FALSE	R
		DBPROP_MULTIPLERESULTS	DBPROPVAL_MR_SUPPORTED	R
		DBPROP_MULTIPLESTORAGEOBJECTS	VARIANT_TRUE	R
		DBPROP_MULTITABLEUPDATE	VARIANT_FALSE	R
		DBPROP_NULLCOLLATION	DBPROPVAL_NC_LOW	R
		DBPROP_OLEOBJECTS	DBPROPVAL_OO_BLOB	R
		DBPROP_ORDERBYCOLUMNSINSELECT	VARIANT_FALSE	R
		DBPROP _OUTPUTPARAMETERAVAILABILITY	DBPROPVAL_OA_ATEXECUTE	R
		DBPROP_PERSISTENTIDTYPE	DBPROPVAL_PT_NAME	R
		DBPROP_PREPAREABORTBEHAVIOR	DBPROPVAL_CB_DELETE	R
		DBPROP_PROCEDURETERM	"PROCEDIMIENTO ALMACENADO"	R

Tabla 24. Propiedades soportadas por IBM OLE DB Provider para DB2 (continuación)

Grupo de propiedades	Conjunto de propiedades	Propiedades	Valor por omisión	R/W
		DBPROP_PROVIDERFRIENDLYNAME	"IBM OLE DB Provider para DB2"	R
		DBPROP_PROVIDERNAME	"IBMDADB2.DLL"	R
		DBPROP_PROVIDEROLEDBVER	"02.7"	R
		DBPROP_PROVIDERVER	N/D	R
		DBPROP_QUOTEIDENTIFIERCASE	DBPROPVAL_IC_SENSITIVE	R
		DBPROP _ROWSETCONVERSIONSONCOMMAND	VARIANT_TRUE	R
		DBPROP_SCHEMATERM	"ESQUEMA"	R
		DBPROP_SCHEMAUSAGE	DBPROPVAL_SU_DML_STATEMENTS | DBPROPVAL_SU_TABLE_DEFINITION | DBPROPVAL_SU_INDEX_DEFINITION | DBPROPVAL_SU_PRIVILEGE_DEFINITION	R
		DBPROP_SQLSUPPORT	DBPROPVAL_SQL_ODBC_EXTENDED | DBPROPVAL_SQL_ESCAPECLAUSES | DBPROPVAL_SQL_ANSI92_ENTRY	R
		DBPROP_SERVERNAME	N/D	R
		DBPROP_STRUCTUREDSTORAGE	DBPROPVAL_SS_ISEQUENTIALSTREAM	R
		DBPROP_SUBQUERIES	DBPROPVAL_SQ_CORRELATEDSUBQUERIES | DBPROPVAL_SQ_COMPARISON | DBPROPVAL_SQ_EXISTS | DBPROPVAL_SQ_IN | DBPROPVAL_SQ_QUANTIFIED |	R
		DBPROP_SUPPORTEDTXNDDL	DBPROPVAL_TC_ALL	R
		DBPROP_SUPPORTEDTXNISOLEVELS	DBPROPVAL_TI_CURSORSTABILITY | DBPROPVAL_TI_READCOMMITTED | DBPROPVAL_TI_READUNCOMMITTED | DBPROPVAL_TI_SERIALIZABLE |	R
		DBPROP_SUPPORTEDTXNISORETAIN	DBPROPVAL_TR_COMMIT_DC | DBPROPVAL_TR_ABORT_NO |	R
		DBPROP_TABLETERM	"TABLA"	R
		DBPROP_USERNAME	N/D	R
Inicialización	DBPROPSET_DBINIT	DBPROP_AUTH_PASSWORD	N/D	R/W
		DBPROP_AUTH_PERSIST _SENSITIVE_AUTHINFO	VARIANT_FALSE	R/W
		DBPROP_AUTH_USERID	N/D	R/W
		DBPROP_INIT_DATASOURCE	N/D	R/W
		DBPROP_INIT_HWND	N/D	R/W
		DBPROP_INIT_MODE	DB_MODE_READWRITE	R/W
		DBPROP_INIT_OLEDBSERVICES	0xFFFFFFFF	R/W
		DBPROP_INIT_PROMPT	DBPROMPT_NOPROMPT	R/W
		DBPROP_INIT_PROVIDERSTRING	N/D	R/W
Conjunto de filas	DBPROPSET_ROWSET	DBPROP_ABORTPRESERVE	VARIANT_FALSE	R
		DBPROP_ACCESSORDER	DBPROPVAL_AO_RANDOM	R
		DBPROP_BLOCKINGSTORAGEOBJECTS	VARIANT_FALSE	R
		DBPROP_BOOKMARKS	VARIANT_FALSE	R/W
		DBPROP_BOOKMARKSKIPPED	VARIANT_FALSE	R
		DBPROP_BOOKMARKTYPE	DBPROPVAL_BMK_NUMERIC	R
		DBPROP_CACHEDEFERRED	VARIANT_FALSE	R/W
		DBPROP_CANFETCHBACKWARDS	VARIANT_FALSE	R/W
		DBPROP_CANHOLDROWS	VARIANT_FALSE	R
		DBPROP_CANSCROLLBACKWARDS	VARIANT_FALSE	R/W
		DBPROP_CHANGEINSERTEDROWS	VARIANT_FALSE	R
		DBPROP_COMMITPRESERVE	VARIANT_TRUE	R/W
		DBPROP_COMMANDTIMEOUT	0	R/W
		DBPROP_DEFERRED	VARIANT_FALSE	R
		DBPROP_IAccessor	VARIANT_TRUE	R
		DBPROP_IColumnsInfo	VARIANT_TRUE	R
		DBPROP_IColumnsRowset	VARIANT_TRUE	R/W
		DBPROP_IConvertType	VARIANT_TRUE	R
		DBPROP_IMultipleResults	VARIANT_FALSE	R/W
		DBPROP_IRowset	VARIANT_TRUE	R
		DBPROP_IRowChange	VARIANT_FALSE	R/W

Tabla 24. Propiedades soportadas por IBM OLE DB Provider para DB2 (continuación)

Grupo de propiedades	Conjunto de propiedades	Propiedades	Valor por omisión	R/W
		DBPROP_IRowsetFind	VARIANT_FALSE	R
		DBPROP_IRowsetIdentity	VARIANT_TRUE	R
		DBPROP_IRowsetInfo	VARIANT_TRUE	R
		DBPROP_IRowsetLocate	VARIANT_FALSE	R/W
		DBPROP_IRowsetScroll	VARIANT_FALSE	R/W
		DBPROP_IRowsetUpdate	VARIANT_FALSE	R
		DBPROP_ISequentialStream	VARIANT_TRUE	R
		DBPROP_ISupportErrorInfo	VARIANT_TRUE	R
		DBPROP_LITERALBOOKMARKS	VARIANT_FALSE	R
		DBPROP_LITERALIDENTITY	VARIANT_TRUE	R
		DBPROP_LOCKMODE	DBPROPVAL_LM_SINGLEROW	R/W
		DBPROP_MAXOPENROWS	32767	R
		DBPROP_MAXROWS	0	R/W
		DBPROP_NOTIFICATIONGRANULARITY	DBPROPVAL_NT_SINGLEROW	R/W
		DBPROP_NOTIFICATION PHASES	DBPROPVAL_NP_OKTODO DBPROPBAL_NP_ABOUTTODO DBPROPVAL_NP_SYNCHAFTER DBPROPVAL_NP_FAILEDTODO DBPROPVAL_NP_DIDEVENT	R
		DBPROP_NOTIFYROWSETRELEASE	DBPROPVAL_NP_OKTODO DBPROPVAL_NP_ABOUTTODO	R
		DBPROP _NOTIFYROWSETFETCHPOSITIONCHANGE	DBPROPVAL_NP_OKTODO DBPROPVAL_NP_ABOUTTODO	R
		DBPROP_NOTIFYCOLUMNSET	DBPROPVAL_NP_OKTODO DBPROPVAL_NP_ABOUTTODO	R
		DBPROP_NOTIFYROWDELETE	DBPROPVAL_NP_OKTODO DBPROPVAL_NP_ABOUTTODO	R
		DBPROP_NOTIFYROWINSERT	DBPROPVAL_NP_OKTODO DBPROPVAL_NP_ABOUTTODO	R
		DBPROP_ORDEREDBOOKMARKS	VARIANT_FALSE	R
		DBPROP_OTHERINSERT	VARIANT_FALSE	R
		DBPROP_OTHERUPDATEDELETE	VARIANT_FALSE	R/W
		DBPROP_OWNINSERT	VARIANT_FALSE	R
		DBPROP_OWNUPDATEDELETE	VARIANT_FALSE	R
		DBPROP_QUICKRESTART	VARIANT_FALSE	R/W
		DBPROP_REMOVEDELETED	VARIANT_FALSE	R/W
		DBPROP_ROWTHREADMODEL	DBPROPVAL_RT_FREETHREAD	R
		DBPROP_SERVERCURSOR	VARIANT_TRUE	R
		DBPROP_SERVERDATAONINSERT	VARIANT_FALSE	R
		DBPROP_UNIQUEROWS	VARIANT_FALSE	R/W
		DBPROP_UPDATABILITY	0	R/W
Conjunto de filas	DBPROPSET_DB2ROWSET	DBPROP_ISLONGMINLENGTH	32000	R/W
Sesión	DBPROPSET_SESSION	DBPROP_SESS_AUTOCOMMITISOLEVELS	DBPROPVAL_TI_CURSORSTABILITY	R/W

Conexiones a fuentes de datos mediante IBM OLE DB Provider

Los siguientes ejemplos muestran cómo conectar con una fuente de datos DB2® mediante IBM® OLE DB Provider para DB2:

Ejemplo 1: Aplicación Visual Basic que utiliza ADO:

```
Dim db As ADODB.Connection
Set db = New ADODB.Connection
db.Provider = "IBMDADB2"
db.CursorLocation = adUseClient
...
```

Ejemplo 2: Aplicación C/C++ que utiliza IDBPromptInitialize y Data Links:

```
// Crear DataLinks
hr = CoCreateInstance (
     CLSID_DataLinks,
     NULL,
     CLSCTX_INPROC_SERVER,
     IID_IDBPromptInitialize,
     (void**)&pIDBPromptInitialize);

// Invocar la UI DataLinks para seleccionar el proveedor y la fuente de datos
hr = pIDBPromptInitialize->PromptDataSource (
     NULL,
     GetDesktopWindow(),
     DBPROMPTOPTIONS_PROPERTYSHEET,
     0,
     NULL,
     NULL,
     IID_IDBInitialize,
     (IUnknown**)&pIDBInitialize);
```

Ejemplo 3: Aplicación C/C++ que utiliza IDataInitialize y el componente de servicio:

```
hr = CoCreateInstance (
     CLSID_MSDAINITIALIZE,
     NULL,
     CLSCTX_INPROC_SERVER,
     IID_IDataInitialize,
     (void**)&pIDataInitialize);

hr = pIDataInitialize->CreateDBInstance(
     CLSID_IBMDADB2, // ClassID de IBMDADB2
     NULL,
     CLSCTX_INPROC_SERVER,
     NULL,
     IID_IDBInitialize,
     (IUnknown**)&pIDBInitialize);
```

Aplicaciones ADO

Las siguientes secciones describen consideraciones sobre aplicaciones ADO.

Palabras clave de series de conexión de ADO

Para especificar palabras clave de series de conexión de ADO (Objetos de datos ActiveX), especifique la palabra clave con el formato `palabra clave=`*`valor`* en la serie (conexión) del proveedor. Delimite varias palabras clave con un punto y coma (;).

La tabla siguiente describe las palabras clave a las que da soporte IBM® OLE DB Provider para DB2®:

Tabla 25. Palabras clave soportadas por IBM OLE DB Provider para DB2

Palabra clave	Valor	Significado
DSN	Nombre del alias de la base de datos	El alias de la base de datos DB2 en el directorio de bases de datos.
UID	ID de usuario	El ID de usuario que se utiliza para conectar con el servidor DB2.
PWD	Contraseña de UID	Contraseña correspondiente al ID de usuario utilizado para conectar con el servidor DB2.

Hay otras palabras clave de configuración de CLI de DB2 que también afectan al comportamiento de IBM OLE DB Provider.

Información relacionada:

- "CLI/ODBC configuration keywords listing by category" en la publicación *CLI Guide and Reference, Volume 1*

Conexiones con fuentes de datos con aplicaciones ADO Visual Basic

Para conectar con una fuente de datos DB2® mediante IBM® OLE DB Provider para DB2, especifique el nombre de proveedor IBMDADB2.

Conceptos relacionados:

- "Conexiones a fuentes de datos mediante IBM OLE DB Provider" en la página 255

Tareas relacionadas:

- "Creación de aplicaciones ADO con Visual Basic" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Cursores desplazables actualizables en aplicaciones ADO

IBM® OLE DB Provider para DB2® da soporte nativo a cursores de sólo lectura, de sólo avance, desplazables y de sólo lectura y a cursores actualizables. Una aplicación ADO que desea acceder a cursores desplazables actualizables puede establecer la ubicación del cursor en `adUseClient` o `adUseServer`. Si establece la ubicación del cursor en `adUseServer`, hace que el cursor se materialice en el servidor.

Limitaciones de las aplicaciones ADO

A continuación se describen las limitaciones de las aplicaciones ADO:

- Las aplicaciones ADO que llaman a procedimientos almacenados deben crear y vincular de forma explícita sus parámetros. El método `Parameters.Refresh` para generar parámetros automáticamente no está soportado para DB2 Server para VSE y VM.
- No se da soporte a valores por omisión de parámetros.
- Al insertar una fila nueva utilizando un cursor desplazable en el lado del servidor, utilice el método AddNew() con los argumentos Fieldlist y Values. Esto es más eficaz que invocar AddNew() sin argumentos a continuación de llamadas Update() para cada columna. Cada llamada AddNew() y Update() es una petición independiente para el servidor y por tanto, es menos eficaz que una única llamada a AddNew().
- Las filas recién insertadas no pueden actualizarse con un cursor desplazable en el extremo del servidor.
- Las tablas con datos largos, LOB, o columnas Datalink no pueden actualizarse al utilizar un cursor desplazable en el extremo del servidor.

Soporte de IBM OLE DB Provider de propiedades y métodos ADO

IBM OLE DB Provider da soporte a los siguientes métodos y propiedades ADO:

Tabla 26. Métodos y propiedades ADO soportados por IBM OLE DB Provider para DB2

ADO	Método/Propiedad	Interfaz/Propiedad OLE DB	Soporte de IBM OLE DB
Métodos de mandato	Cancel	ICommand	Sí
	CreateParameter		Sí
	Execute		Sí
Propiedades de mandato	ActiveConnection	(Específico de ADO)	
	Command Text	ICommandText	Sí
	Command Timeout	ICommandProperties::SetProperties DBPROP_COMMANDTIMEOUT	Sí
	CommandType	(Específico de ADO)	
	Prepared	ICommandPrepare	Sí
	State	(Específico de ADO)	
Grupo de mandatos	Parameters	ICommandWithParameter DBSCHEMA _PROCEDURE_PARAMETERS	Sí
	Properties	ICommandProperties IDBProperties	Sí
Métodos de conexión	BeginTrans CommitTrans RollbackTrans	ITransactionLocal	Sí (pero no anidado) Sí (pero no anidado) Sí (pero no anidado)
	Execute	ICommand IOpenRowset	Sí
	Open	IDBCreateSession IDBInitialize	Sí
	OpenSchema adSchemaColumnPrivileges adSchemaColumns adSchemaForeignKeys adSchemaIndexes adSchemaPrimaryKeys adSchemaProcedureParam adSchemaProcedures adSchemaProviderType adSchemaStatistics adSchemaTablePrivileges adSchemaTables	IDBSchemaRowset	 Sí Sí Sí Sí Sí Sí Sí Sí Sí Sí Sí
	Cancel		Sí
Propiedades de conexión	Attributes adXactCommitRetaining adXactRollbackRetaining	ITransactionLocal	 Sí Sí
	CommandTimeout	ICommandProperties DBPROP_COMMAND_TIMEOUT	Sí
	ConnectionString	(Específico de ADO)	
	ConnectionTimeout	IDBProperties DBPROP_INIT_TIMEOUT	No

Tabla 26. Métodos y propiedades ADO soportados por IBM OLE DB Provider para DB2 (continuación)

ADO	Método/Propiedad	Interfaz/Propiedad OLE DB	Soporte de IBM OLE DB
	CursorLocation: adUseClient adUseNone adUseServer	 (Utilizar Servicio de cursor de OLE DB) (No utilizado)	 Sí No Sí
	DefaultDataBase	IDBProperties DBPROP_CURRENTCATALOG	No
	IsolationLevel	ITransactionLocal DBPROP_SESS _AUTOCOMMITISOLEVELS	Sí
	Mode adModeRead adModeReadWrite adModeShareDenyNone adModeShareDenyRead adModeShareDenyWrite adModeShareExclusive adModeUnknown adModeWrite	IDBProperties DBPROP_INIT_MODE	 No Sí NoNoNoNoNo No
	Provider	ISourceRowset::GetSourceRowset	Sí
	State	(Específico de ADO)	
	Version	(Específico de ADO)	
Grupos de conexiones	Errors	IErrorRecords	Sí
	Properties	IDBProperties	Sí
Propiedades de error	Description NativeError Number Source SQLState	IErrorRecords	Sí Sí Sí Sí Sí
	HelpContext HelpFile		No No
Métodos de campo	AppendChunk GetChunk	ISequentialStream	Sí Sí
Propiedades de campo	Actual Size	IAccessor IRowset	Sí
	Attributes DataFormat DefinedSize Name NumericScale Precision Type	IColumnInfo	 Sí Sí Sí Sí Sí Sí
	OriginalValue	IRowsetUpdate	Sí (Servicio de cursor)
	UnderlyingValue	IRowsetRefresh IRowsetResynch	Sí (Servicio de cursor) Sí (Servicio de cursor)
	Value	IAccessor IRowset	Sí
Grupo de campos	Properties	IDBProperties IRowsetInfo	Sí

Tabla 26. Métodos y propiedades ADO soportados por IBM OLE DB Provider para DB2 (continuación)

ADO	Método/Propiedad	Interfaz/Propiedad OLE DB	Soporte de IBM OLE DB
Métodos de parámetro	AppendChunk	ISequentialStream	Sí
	Attributes Direction Name NumericScale Precision Scale Size Type	ICommandWithParameter DBSCHEMA _PROCEDURE_PARAMETERS	 Sí No Sí Sí Sí Sí Sí
	Value	IAccessor ICommand	Sí
Grupo de parámetros	Properties		Sí
Métodos RecordSet	AddNew	IRowsetChange	Sí
	Cancel		Sí
	CancelBatch	IRowsetUpdate::Undo	Sí (Servicio de cursor)
	CancelUpdate		Sí (Servicio de cursor)
	Clone	IRowsetLocate	Sí
	Close	IAccessor IRowset	Sí
	CompareBookmarks		No
	Delete	IRowsetChange	Sí
	GetRows	IAccessor IRowset	Sí
	Move	IRowset IRowsetLocate	Sí
	MoveFirst	IRowset IRowsetLocate	Sí
	MoveNext	IRowset IRowsetLocate	Sí
	MoveLast	IRowsetLocate	Sí
	MovePrevious	IRowsetLocate	Sí
	NextRecordSet	IMultipleResults	Sí
	Open	ICommand IOpenRowset	Sí
	Requery	ICommand IOpenRowset	Sí
	Resync	IRowsetRefresh	Sí (Servicio de cursor)
	Supports	IRowsetInfo	Sí
	Update UpdateBatch	IRowsetChange IRowsetUpdate	Sí Sí (Servicio de cursor)
Propiedades de RecordSet	AbsolutePage	IRowsetLocate IRowsetScroll	Sí Sí[1]
	AbsolutePosition	IRowsetLocate IRowsetScroll	Sí Sí[1]
	ActiveConnection	IDBCreateSession IDBInitialize	Sí
	BOF	(Específico de ADO)	

Tabla 26. Métodos y propiedades ADO soportados por IBM OLE DB Provider para DB2 (continuación)

ADO	Método/Propiedad	Interfaz/Propiedad OLE DB	Soporte de IBM OLE DB
	Bookmark	IAccessor IRowsetLocate	Sí
	CacheSize	cRows in IRowsetLocate IRowset	Sí
	CursorType adOpenDynamic adOpenForwardOnly adOpenKeySet adOpenStatic	ICommandProperties	 No Sí Sí Sí
	EditMode	IRowsetUpdate	Sí (Servicio de cursor)
	EOF	(Específico de ADO)	
	Filter	IRowsetLocate IRowsetView IRowsetUpdate IViewChapter IViewFilter	No
	LockType	ICommandProperties	Sí
	MarshallOption		No
	MaxRecords	ICommandProperties IOpenRowset	Sí
	PageCount	IRowsetScroll	Sí[1]
	PageSize	(Específico de ADO)	
	Sort	(Específico de ADO)	
	Source	(Específico de ADO)	
	State	(Específico de ADO)	
	Status	IRowsetUpdate	Sí (Servicio de cursor)
Notas: 1. Los valores a devolver son aproximaciones. Las filas suprimidas no se pasarán por alto.			
Grupo de RecordSet	Fields	IColumnInfo	Sí
	Properties	IDBProperties IRowsetInfo::GetProperties	Sí

Aplicaciones C y C++

Las secciones siguientes describen consideraciones sobre aplicaciones C y C++.

Compilación y enlace de aplicaciones C/C++ e IBM OLE DB Provider

Las aplicaciones C/C++ que utilizan la constante CLSID_IBMDADB2 deben incluir el archivo `ibmdadb2.h`, que se encuentra en el directorio `SQLLIB\include`. Estas aplicaciones deben definir DBINITCONSTANTS antes de la sentencia include. El siguiente ejemplo muestra la secuencia correcta de sentencias:

```
#define DBINITCONSTANTS
#include "ibmdadb2.h"
```

Conexiones con fuentes de datos en aplicaciones C/C++ mediante IBM OLE DB Provider

Para conectar con una fuente de datos DB2® mediante IBM® OLE DB Provider para DB2 en una aplicación C/C++, utilice una de las dos interfaces principales de OLE DB: `IDBPromptInitialize` o `IDataInitialize`, o bien puede invocar la API CoCreateInstance de COM. El componente OLE DB Service expone la interfaz `IDataInitialize` y el componente Data Links expone la interfaz `IDBPromptInitialize`.

Conceptos relacionados:

- "Conexiones a fuentes de datos mediante IBM OLE DB Provider" en la página 255

Tareas relacionadas:

- "Creación de aplicaciones ADO con Visual C++" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Transacciones distribuidas MTS y COM+

Las secciones siguientes describen consideraciones sobre las transacciones distribuidas MTS y COM+.

Soporte de transacciones distribuidas MTS y COM+ e IBM OLE DB Provider

Las aplicaciones OLE DB que se ejecutan en un entorno Microsoft® Transaction Server (MTS) en Windows® NT o en un entorno Component Services (COM+) en Windows 2000 pueden utilizar la interfaz `ITransactionJoin` para participar en transacciones distribuidas con varios servidores de bases de datos DB2® Universal Database, de sistema principal y iSeries, así como otros gestores de recursos que cumplan con las especificaciones MTS/COM+.

Requisitos previos:

Para poder utilizar el soporte de transacciones distribuidas MTS o COM+ que ofrece IBM® OLE DB Provider para DB2, asegúrese de que el servidor cumple los siguientes requisitos previos.

Nota: Estos requisitos sólo afectan a la máquina Windows en la que está instalado el cliente DB2.

- Windows NT® con MTS al nivel de Versión 2.0 con Microsoft Hotfix 0772 o posterior

  MTS Versión 2.0 para Windows NT está disponible como parte de Windows NT 4.0 Option Pack. Puede bajar el Option Pack de:

  `http://www.microsoft.com/ntserver/nts/downloads/recommended/NT4OptPk/`

- Windows 2000 con Service Pack 3 o posterior

Conceptos relacionados:

- "Microsoft Transaction Server (MTS) y Microsoft Component Services (COM+) como gestor de transacciones" en la página 691
- "Soporte ligeramente agrupado con Microsoft Component Services (COM+)" en la página 693

Habilitación del soporte de MTS en DB2 Universal Database para aplicaciones C/C++

Para ejecutar una aplicación C o C++ en la modalidad transaccional MTS o COM+, puede crear la instancia de fuente de datos IBMDADB2 mediante la interfaz `DataLink`. También podría utilizar `CoCreateInstance`, obtener un objeto de sesión y utilizar `JoinTransaction`. Consulte la descripción sobre cómo conectar una aplicación C o C++ a una fuente de datos para obtener más información.

Para ejecutar una aplicación ADO en modalidad transaccional MTS o COM+, consulte la descripción sobre cómo conectar una aplicación C o C++ a una fuente de datos.

Para utilizar un componente de un paquete MTS o COM+ en modalidad transaccional, establezca para la propiedad Transactions del componente uno de los siguientes valores:

- "Required"
- "Required New"
- "Supported"

Para obtener información sobre estos valores, consulte la documentación de MTS.

Conceptos relacionados:

- "Microsoft Transaction Server (MTS) y Microsoft Component Services (COM+) como gestor de transacciones" en la página 691
- "Soporte ligeramente agrupado con Microsoft Component Services (COM+)" en la página 693

Capítulo 12. OLE DB .NET Data Provider

OLE DB .NET Data Provider 265
Restricciones de OLE DB .NET Data Provider . . 266
Agrupación de conexiones en aplicaciones de OLE DB .NET Data Provider 270
Columnas de tiempo en aplicaciones de OLE DB .NET Data Provider 271
Objetos ADORecordset en aplicaciones de OLE DB .NET Data Provider 272

OLE DB .NET Data Provider

OLE DB .NET Data Provider utiliza el Controlador IBM® DB2® OLE DB, que se especifica como `IBMDADB2` en un objeto `ConnectionString`. Las palabras clave de la serie de conexión soportadas por OLE DB .NET Data Provider son las misma que las soportadas por IBM OLE DB Provider para DB2. Además, OLE DB .NET Data Provider tiene las mismas restricciones que IBM DB2 OLE DB Provider. Existen restricciones adicionales para OLE DB .NET Data Provider, que se describen en el tema: Restricciones de OLE DB .NET Data Provider.

Para utilizar OLE DB .NET Data Provider, debe tener instalado .NET Framework Versión 1.1.

Para DB2 Universal Database™ para AS/400® e iSeries™, es necesario el siguiente arreglo de programa en el servidor: APAR ii13348.

Estas son las palabras clave de conexión soportadas para OLE DB .NET Data Provider:

Tabla 27. Palabras clave de **ConnectionString** *para OLE DB .NET Data Provider*

Palabra clave	Value	Significado
`PROVIDER`	`IBMDADB2`	Especifica el IBM OLE DB Provider para DB2 (obligatorio)
`DSN` o `Data Source`	alias de base de datos	El alias de la base de datos DB2 tal como está catalogado en el directorio de bases de datos.
`UID`	ID de usuario	El ID de usuario que se utiliza para conectar con el servidor DB2
`PWD`	contraseña	La contraseña correspondiente al ID de usuario utilizado para conectar con el servidor DB2

El ejemplo siguiente crea una conexión `OleDbConnection` para conectar con la base de datos SAMPLE:

```
[Visual Basic .NET]
Dim con As New OleDbConnection("Provider=IBMDADB2;" +
    "Data Source=sample;UID=userid;PWD=password;")
con.Open()

[C#]
OleDbConnection con = new OleDbConnection("Provider=IBMDADB2;" +
                "Data Source=sample;UID=userid;PWD=password;" );
con.Open()
```

Restricciones de OLE DB .NET Data Provider

La tabla siguiente muestra las restricciones de utilización de IBM OLE DB .NET Data Provider:

Tabla 28. Restricciones de IBM OLE DB .NET Data Provider

Clase o función	Descripción de la restricción	Servidores DB2 afectados
Corrientes de caracteres ASCII	No puede utilizar corrientes de caracteres ASCII con `OleDbParameters` si utiliza `DbType.AnsiString` o `DbType.AnsiStringFixedLength`. OLE DB .NET Data Provider emitirá la excepción siguiente: `"Specified cast is not valid" (la conversión de datos especificada no es válida)` **Corrección:** Utilice `DbType.Binary` en lugar de utilizar `DbType.AnsiString` o `DbType.AnsiStringFixedLength`.	Todos
`ADORecord`	`ADORecord` no está soportado.	Todos

`ADORecordSet` y `Timestamp`	Tal como está documentado en MSDN, la variable `ADORecordSet` toma el valor 1 segundo. Como consecuencia, las fracciones de segundo se pierden cuando una columna `Timestamp` se almacena en un `ADORecordSet`. Similarmente, después de llenar un `DataSet` de un `ADORecordSet`, las columnas `Timestamp` de `DataSet` no tendrán fracciones de segundo.
	
	Solución:
	
	Esta solución solo es efectiva para DB2 Universal Database para Linux, UNIX y Windows, Versión 8.1, FixPak 4 o posterior. Para evitar la pérdida de fracciones de segundo, puede definir la siguiente palabra clave de CLI:
	`MAPTIMESTAMPDESCRIBE = 2`
	
	Esta palabra clave describe `Timestamp` como WCHAR(26). Para definir la palabra clave, ejecute el mandato siguiente desde una ventana de mandatos de DB2:
	`db2 update cli cfg for section common using MAPTIMESTAMPDESCRIBE 2`
		Todos

`Capítulos`	Los capítulos no están soportados.	Todos
Información de claves	OLE DB .NET Data Provider no puede obtener información de claves al abrir un `IDataReader` al mismo tiempo.	DB2 para VM/VSE

Tabla 28. Restricciones de IBM OLE DB .NET Data Provider (continuación)

Clase o función	Descripción de la restricción	Servidores DB2 afectados
Información de claves procedente de procedimientos almacenados	OLE DB .NET Data Provider puede recuperar información de claves referente a un conjunto de resultados devuelto por un procedimiento almacenado solo desde DB2 Universal Database para Linux, UNIX y Windows. Esto es debido a que los servidores DB2 para plataformas que no sean Linux, UNIX y Windows no devuelven información descriptiva ampliada para los conjuntos de resultados abiertos en el procedimiento almacenado. Para recuperar información de claves de un conjunto de resultados devuelto por un procedimiento almacenado en DB2 Universal Database para Linux, UNIX y Windows, es necesario definir la siguiente variable de registro en el servidor DB2: `db2set DB2_APM_PERFORMANCE=8` El definir esta variable de registro de DB2 del servidor hará que los metadatos del conjunto de resultados estén disponibles en el servidor durante más tiempo, lo cual permite que OLE DB recupere satisfactoriamente la información de claves. Sin embargo, dependiendo de la carga de trabajo del servidor, puede que los metadatos no estén disponibles el tiempo suficiente para que OLE DB Provider consulte la información. Por tanto, no es seguro que la información de claves esté siempre disponible para los conjuntos de resultados devueltos por un procedimiento almacenado. Para recuperar cualquier información de claves sobre una sentencia CALL, la aplicación debe ejecutar la sentencia CALL. La invocación de `OleDbDataAdapter.FillSchema()` o `OleDbCommand.ExecuteReader(CommandBehavior.SchemaOnly | CommandBehavior.KeyInfo)` no ejecutará realmente la llamada de procedimiento almacenado. Por tanto, el usuario no obtiene ninguna información de claves para el conjunto de resultados que debe ser devuelto por el procedimiento almacenado.	Todos

Información de claves de sentencias de SQL de proceso por lotes	Cuando se utilizan sentencias de SQL de proceso por lotes que devuelven varios resultados, el método `FillSchema()` intenta recuperar información de esquemas solo para la primera sentencia de SQL contenida en la lista de sentencias de SQL de proceso por lotes. Si esta sentencia no devuelve un conjunto de resultados, no se crea ninguna tabla. Por ejemplo:
	`[C#]`
	`cmd.CommandText = "INSERT INTO ORG(C1) VALUES(1000); SELECT C1 FROM ORG;";`
	`da = new OleDbDataAdapter(cmd);`
	`da.FillSchema(ds, SchemaType.Source);`
	
	No se creará ninguna tabla en el archivo, pues la primera sentencia de SQL del lote es una sentencia INSERT, que no devuelve un conjunto de resultados.
		Todos

Tabla 28. Restricciones de IBM OLE DB .NET Data Provider (continuación)

Clase o función	Descripción de la restricción	Servidores DB2 afectados
`OleDbCommandBuilder`	Las sentencias UPDATE, DELETE e INSERT generadas automáticamente por `OleDbCommandBuilder` son incorrectas si la sentencia SELECT contiene cualquier columna de los tipos de datos siguientes: • CLOB • BLOB • DBCLOB • LONG VARCHAR • LONG VARCHAR FOR BIT DATA • LONG VARGRAPHIC Si se conecta a un servidor DB2 que no sea DB2 Universal Database para Linux, Unix y Windows, este problema también es debido a columnas con los tipos de datos siguientes: • VARCHAR[1] • VARCHAR FOR BIT DATA[1] • VARGRAPHIC[1] • REAL • FLOAT o DOUBLE • TIMESTAMP **Notas:** 1. Las columnas con estos tipos de datos son aplicables si están definidas para ser valores VARCHAR mayores que 254 bytes, valores VARCHAR FOR BIT DATA mayores que 254 bytes o valores VARGRAPHICs mayores que 127 bytes. Esta condición solo es válida si se conecta a un servidor DB2 que no sea DB2 Universal Database para Linux, Unix y Windows. `OleDbCommandBuilder` genera sentencias de SQL que utilizan todas las columnas seleccionadas en una comparación de igualdad de la cláusula WHERE, pero los tipos de datos listados anteriormente no se pueden utilizar en una comparación de igualdad. **Nota:** Observe que esta restricción no afectará al método `IDbDataAdapter.Update()` que depende de `OleDbCommandBuilder` para generar automáticamente las sentencias UPDATE, DELETE e INSERT. La operación UPDATE no es efectiva si la sentencia generada contiene cualquiera de los tipos de datos listados anteriormente. **Solución:** Es necesario que elimine explícitamente en la cláusula WHERE de la sentencia de SQL generada todas las columnas que tengan cualquiera de los tipos de datos listados anteriormente. Es recomendable que codifique sus propias sentencias UPDATE, DELETE e INSERT.	Todos
`OleDbCommandBuilder. DeriveParameters`	La distinción entre mayúsculas y minúsculas es importante al utilizar `DeriveParameters()`. El nombre de procedimiento almacenado especificado en `OleDbCommand.CommandText` necesita estar escrito exactamente tal como está almacenado en la tablas de catálogo del sistema DB2. Para ver cómo están almacenados los nombres de procedimiento almacenado, ejecute OpenSchema(OleDbSchemaGuid.Procedures) sin proporcionar la restricción de nombre del procedimiento. Esto devolverá todos los nombres de procedimiento almacenado. Por omisión, DB2 almacena los nombres de procedimiento almacenado en letras mayúsculas, por lo que generalmente deberá especificar el nombre del procedimiento almacenado en mayúsculas.	Todos
`OleDbCommandBuilder. DeriveParameters`	El método `OleDbCommandBuilder.DeriveParameters()` no incluye el parámetro `ReturnValue` en el objeto `OleDbParameterCollection` generado. Por omisión, SqlClient y DB2 .NET Data Provider añaden el parámetro con `ParameterDirection.ReturnValue` al objeto `ParameterCollection` generado.	Todos

Tabla 28. Restricciones de IBM OLE DB .NET Data Provider (continuación)

Clase o función	Descripción de la restricción	Servidores DB2 afectados
`OleDbCommandBuilder. DeriveParameters`	El método `OleDbCommandBuilder.DeriveParameters()` no es efectivo para procedimientos almacenados sobrecargados. Si existen varios procedimientos almacenados denominados "MYPROC", cada uno de los cuales acepta un número diferente de parámetros o tipos diferentes de parámetros, `OleDbCommandBuilder.DeriveParameters()` recupera todos los parámetros de todos los procedimientos almacenados sobrecargados.	Todos
`OleDbCommandBuilder. DeriveParameters`	Si la aplicación no autoriza un procedimiento almacenado con un esquema, `DeriveParameters()` devuelve todos los parámetros para ese nombre de procedimiento. Por tanto, si existen varios esquemas para el mismo nombre de procedimiento, `DeriveParameters()` devuelve todos los parámetros para todos los procedimientos del mismo nombre.	Todos
`OleDbConnection. ChangeDatabase`	El método `OleDbConnection.ChangeDatabase()` no está soportado.	Todos
`OleDbConnection. ConnectionString`	El uso de caracteres no imprimibles, tales como '\b', '\a' o '\O' en la serie de conexión hará que se emita una excepción. Existen restricciones para las palabras clave siguientes: **`Data Source`** La fuente de datos es el nombre de la base de datos, no el servidor. Puede especificar la palabra clave SERVER, pero no será tenida en cuenta por el proveedor IBMDADB2. **`Initial Catalog` y `Connect Timeout`** Estas palabras clave no están soportadas. En general, OLE DB .NET Data Provider no tiene en cuenta ninguna de las palabras clave no reconocidas y no soportadas. Pero si especifica estas palabras clave, se emite la excepción siguiente: `La operación de OLE DB de varios pasos ha generado errores. Examine cada valor de estado de OLE DB, si está disponible. No se ha realizado ninguna acción.` **`ConnectionTimeout`** `ConnectionTimeout` es de solo lectura.	Todos

`OleDbConnection. GetOleDbSchemaTable`	Los valores de restricción distinguen entre mayúsculas y minúsculas, y necesitan estar escritos exactamente tal como están especificados los objetos de base de datos almacenados en las tablas de catálogo del sistema. Por omisión, esos valores se almacenan en mayúsculas.
	
	Por ejemplo, si ha creado una tabla de la manera siguiente:
	`CREATE TABLE abc(c1 SMALLINT)`
	
	DB2 almacenará el nombre de tabla en mayúsculas ("ABC") en el catálogo del sistema. Por tanto, debe utilizar "ABC" como valor de restricción. Por ejemplo:
	`schemaTable = con.GetOleDbSchemaTable(OleDbSchemaGuid.Tables, new object[] { null, null, "ABC", "TABLE" });`
	
	Corrección:
	
	Si necesita que se distinga entre mayúsculas y minúsculas o necesita especificar espacios en blanco en las definiciones de datos, encierre las definiciones entre comillas. Por ejemplo:
	`cmd.CommandText = "create table \"Mi Tabla\"(c1 int)";`
	`cmd.ExecuteNonQuery();`
	`tablename = "\"Mi Tabla\"";`
	`schemaTable = con.GetOleDbSchemaTable(OleDbSchemaGuid.Tables, new object[] { null, null, tablename, "TABLE" });`
		Todos

Tabla 28. Restricciones de IBM OLE DB .NET Data Provider (continuación)

Clase o función	Descripción de la restricción	Servidores DB2 afectados
`OleDbDataAdapter` y `DataColumnMapping`	El nombre de la columna fuente distingue entre mayúsculas y minúsculas. Debe estar escrita exactamente tal como está almacenada en los catálogos de DB2. Por omisión, los nombres de columna se almacenan en mayúsculas. Por ejemplo: `colMap = new DataColumnMapping("EMPNO", "Employee ID");`	Todos
`OleDbDataReader.GetSchemaTable`	OLE DB .NET Data Provider no puede recuperar información descriptiva ampliada procedente de servidores que no devuelven esa clase de información. Si se conecta a un servidor que no da soporte a información descriptiva ampliada (los servidores afectados), las columnas siguientes de la tabla de metadatos devuelta desde `IDataReader.GetSchemaTable()` no son válidas: • `IsReadOnly` • `IsUnique` • `IsAutoIncrement` • `BaseSchemaName` • `BaseCatalogName`	DB2 para OS/390, versión 7 o inferior DB2 para OS/400 DB2 para VM/VSE
Procedimientos almacenados: sin nombres de columna para conjuntos de resultados	El servidor DB2 para OS/390 versión 6.1 no devuelve nombres de columna para conjuntos de resultados devueltos desde un procedimiento almacenado. OLE DB .NET Data Provider correlaciona estas columnas sin nombre con su número de posición ordinal (por ejemplo, ″1″, ″2″ ″3″). Esto es diferente de la correlación documentada en MSDN: `"Columna1", "Columna2", "Columna3"`.	DB2 para OS/390 versión 6.1

Agrupación de conexiones en aplicaciones de OLE DB .NET Data Provider

OLE DB .NET Data Provider agrupa automáticamente las conexiones utilizando el servicio de agrupación de sesiones de OLE DB. Se pueden utilizar argumentos de la serie de conexión para habilitar o inhabilitar servicios de OLE DB, incluida la agrupación de conexiones. Por ejemplo, la siguiente serie de conexión inhabilita la agrupación de sesiones y el enlistamiento automático de transacciones de OLE DB.

```
Provider=IBMDADB2;OLE DB Services=-4;Data Source=SAMPLE;
```

La tabla siguiente describe los atributos de la serie de conexión de ADO que puede utilizar para definir los servicios de OLE DB:

Tabla 29. Definición de servicios de OLE DB mediante atributos de la serie de conexión de ADO

Servicios habilitados	Valor en la serie de conexión
Todos los servicios (valor por omisión)	″OLE DB Services = -1;″
Todos los servicios excepto la agrupación	″OLE DB Services = -2;″
Todos los servicios excepto la agrupación y el enlistamiento automático	″OLE DB Services = -4;″
Todos los servicios excepto el cursor de cliente	″OLE DB Services = -5;″
Todos los servicios excepto el cursor de cliente y la agrupación	″OLE DB Services = -6;″
Ningún servicio	″OLE DB Services = 0;″

Para obtener más información sobre la agrupación de sesiones o de recursos en OLE DB, así como sobre la inhabilitación del servicio de agrupación mediante la

modificación de los valores por omisión del servicio proveedor de OLE, consulte el manual de consulta del programador de OLE DB, que se encuentra en la biblioteca de MSDN situada en http://msdn.microsoft.com/library.

Columnas de tiempo en aplicaciones de OLE DB .NET Data Provider

Las secciones siguientes describen implementar columnas de tiempo en aplicaciones de OLE DB .NET Data Provider.

Inserción marcadores de parámetros:

Por ejemplo, suponga que desea insertar un valor de tiempo en una columna de Tiempo:

```
command.CommandText = "insert into mytable(c1) values( ? )";
```

donde la columna c1 es una columna de Tiempo. He aquí dos métodos para vincular un valor de tiempo al marcador de parámetros:

Utilizando `OleDbParameter.OleDbType = OleDbType.DBTime`

Debido a que OleDbType.DBTime se correlaciona con un objeto TimeSpan, debe proporcionar un objeto TimeSpan como valor de parámetro. El valor de parámetro no puede ser un objeto String ni DateTime; debe ser un objeto TimeSpan. Por ejemplo:

```
p1.OleDbType = OleDbType.DBTime;
p1.Value = TimeSpan.Parse("0.11:20:30");
rowsAffected = cmd.ExecuteNonQuery();
```

El formato de TimeSpan es una serie de caracteres de la forma: ″[-]d.hh:mm:ss.ff″, tal como está documentado en el documentación de MSDN.

Utilizando `OleDbParameter.OleDbType = OleDbType.DateTime`

Esta especificación hará que OLE DB .NET Data Provider convierta el valor de parámetro en un objeto DateTime, en lugar de un objeto TimeSpan. Como consecuencia, el valor de parámetro puede ser cualquier serie/objeto válido que se pueda convertir en un objeto DateTime. Esto significa que valores tales como ″11:20:30″ serán efectivos. El valor también puede ser un objeto DateTime. El valor no puede ser un objeto TimeSpan, pues no se puede convertir en un objeto DateTime. TimeSpan no implementa IConvertible.

Por ejemplo:

```
p1.OleDbType = OleDbType.DBTimeStamp;
p1.Value = "11:20:30";
rowsAffected = cmd.ExecuteNonQuery();
```

Recuperación:

Para recuperar una columna de tiempo necesita utilizar el método `IDataRecord.GetValue()` o `OleDbDataReader.GetTimeSpan()`.

Por ejemplo:

```
TimeSpan ts1 = ((OleDbDataReader)reader).GetTimeSpan( 0 );
TimeSpan ts2 = (TimeSpan) reader.GetValue( 0 );
```

Objetos ADORecordset en aplicaciones de OLE DB .NET Data Provider

Las consideraciones siguientes tratan sobre la utilización de objetos ADORecordset.

- La clase adDBTime de tipo ADO se correlaciona con la clase DateTime de .NET Framework. OleDbType.DBTime corresponde a un objeto TimeSpan.
- No puede asignar un objeto TimeSpan al campo Time de un objeto ADORecordset. Esto es debido a que el campo Time del objeto ADORecordset espera recibir un objeto DateTime. Cuando asigna un objeto TimeSpan a un objeto ADORecordset, recibe el mensaje siguiente:

  ```
  La signatura de tipo del método no es compatible con Interop.
  ```

  Solo puede llenar el campo Time con un objeto DateTime, o con un objeto String que se pueda convertir en un objeto DateTime.
- Cuando llena un DataSet con un ADORecordset utilizando OleDbDataAdapter, el campo Time de ADORecordset se convierte en una columna TimeSpan de DataSet.
- Los Recordsets no almacenan claves primarias ni restricciones. Por tanto, no se añade ninguna información de claves al llenar un DataSet a partir de un Recordset utilizando MissingSchemaAction.AddWithKey.

Capítulo 13. ODBC .NET Data Provider

ODBC .NET Data Provider . . . . . 273 | Restricciones de ODBC .NET Data Provider . . . 273

ODBC .NET Data Provider

ODBC .NET Data Provider efectúa llamadas de ODBC a una fuente de datos DB2® utilizando el Controlador de CLI de DB2. Por tanto, las palabras de serie de conexión soportadas por ODBC .NET Data Provider son las mismas que las soportadas por el controlador de CLI de DB2. Además, ODBC DB .NET Data Provider tiene las mismas restricciones que el controlador de CLI de DB2. Existen restricciones adicionales para ODBC .NET Data Provider, que se describen en el tema: Restricciones de ODBC .NET Data Provider.

Para utilizar ODBC .NET Data Provider, debe tener instalado .NET Framework Versión 1.1. Para DB2 Universal Database para AS/400® e iSeries™, es necesario el siguiente arreglo de programa en el servidor: APAR II13348.

Estas son las palabras clave de conexión soportadas para ODBC .NET Data Provider:

Tabla 30. Palabras clave de **ConnectionString** *para ODBC .NET Data Provider*

Palabra clave	Value	Significado
DSN	alias de base de datos	El alias de la base de datos DB2 tal como está catalogado en el directorio de bases de datos.
UID	ID de usuario	El ID de usuario que se utiliza para conectar con el servidor DB2
PWD	contraseña	La contraseña correspondiente al ID de usuario utilizado para conectar con el servidor DB2

El ejemplo siguiente crea una conexión `OdbcConnection` para conectar con la base de datos SAMPLE:

```
[Visual Basic .NET]
Dim con As New OdbcConnection("DSN=sample;UID=userid;PWD=password;")
con.Open()

[C#]
OdbcConnection con = new OdbcConnection("DSN=sample;UID=userid;PWD=password;");
con.Open()
```

Restricciones de ODBC .NET Data Provider

La tabla siguiente muestra las restricciones de utilización de IBM ODBC .NET Data Provider:

Tabla 31. Restricciones de IBM ODBC .NET Data Provider

Clase o función	Descripción de la restricción	Servidores DB2 afectados
Corrientes de caracteres ASCII	No puede utilizar corrientes de caracteres ASCII con `OdbcParameters` si utiliza `DbType.AnsiString` o `DbType.AnsiStringFixedLength`. ODBC .NET Data Provider emitirá la excepción siguiente: `"Specified cast is not valid" (la conversión de tipo de datos especificada no es válida)` **Corrección:** Utilice `DbType.Binary` en lugar de utilizar `DbType.AnsiString` o `DbType.AnsiStringFixedLength`.	Todos

`Command.Prepare`	Antes de ejecutar un mandato (`Command.ExecuteNonQuery` o `Command.ExecuteReader`), debe ejecutar explícitamente `OdbcCommand.Prepare()` si `CommandText` ha cambiado desde la última preparación. Si no ejecuta `OdbcCommand.Prepare()` de nuevo, ODBC .NET Data Provider ejecutará el `CommandText` preparado anteriormente.
	
	Por ejemplo:
	`[C#]`
	`command.CommandText="select CLOB('ABC') from table1";`
	`command.Prepare();`
	`command.ExecuteReader();`
	`command.CommandText="select CLOB('XYZ') from table2";`
	`command.ExecuteReader(); // Esto finaliza ejecutando de nuevo`
	`// la primera sentencia`
		Todos

CommandBehavior. SequentialAccess	Si utiliza `IDataReader.GetChars()` para leer con un lector creado con `CommandBehavior.SequentialAccess`, debe asignar un almacenamiento intermedio que sea lo suficiente grande para contener la columna completa. De lo contrario, recibirá la excepción siguiente:
	`Requested range extends past the end of the array.`
	`at System.Runtime.InteropServices.Marshal.Copy(Int32 source,`
	`Char[] destination, Int32 startIndex, Int32 length)`
	`at System.Data.Odbc.OdbcDataReader.GetChars(Int32 i,`
	`Int64 dataIndex, Char[] buffer, Int32 bufferIndex, Int32 length)`
	`at OleRestrict.TestGetCharsAndBufferSize(IDbConnection con)`
	
	El ejemplo siguiente muestra cómo asignar un almacenamiento intermedio apropiado:
	`CREATE TABLE myTable(c0 int, c1 CLOB(10K))`
	`SELECT c1 FROM myTable;`
	
	`[C#]`
	`cmd.CommandText = "SELECT c1 from myTable";`
	`IDataReader reader = cmd.ExecuteReader(CommandBehavior.SequentialAccess);`
	
	`Int32 iChunkSize = 10;`
	`Int32 iBufferSize = 10;`
	`Int32 iFieldOffset = 0;`
	
	`Char[] buffer = new Char[ iBufferSize ];`
	
	`reader.Read();`
	`reader.GetChars(0, iFieldOffset, buffer, 0, iChunkSize);`
	
	La llamada a `GetChars()` emitirá la excepción siguiente:
	`"Requested range extends past the end of the array"`
	
	Para evitar que `GetChars()` emita la excepción anterior, asigne a `BufferSize` un valor igual al tamaño de la columna, de esta manera:
	`Int32 iBufferSize = 10000;`
	
	Observe que el valor de 10000 para `iBufferSize` corresponde al valor de 10K asignado a la columna CLOB `c1`.
		Todos

Tabla 31. Restricciones de IBM ODBC .NET Data Provider (continuación)

Clase o función	Descripción de la restricción	Servidores DB2 afectados

CommandBehavior. SequentialAccess	ODBC .NET Data Provider emite la excepción siguiente cuando no existen más datos para leer al utilizar `OdbcDataReader.GetChars()`:
	`NO_DATA - no error information available`
	`at System.Data.Odbc.OdbcConnection.HandleError(HandleRef hrHandle, SQL_HANDLE hType, RETCODE retcode)`
	`at System.Data.Odbc.OdbcDataReader.GetData(Int32 i, SQL_C sqlctype, Int32 cb)`
	`at System.Data.Odbc.OdbcDataReader.GetChars(Int32 i, Int64 dataIndex, Char[] buffer, Int32 bufferIndex, Int32 length)`
		Todos

CommandBehavior. SequentialAccess	No podrá utilizar tamaños de bloques de datos grandes, tales como el representado por el valor 5000, al utilizar `OdbcDataReader.GetChars()`. Cuando intente utilizar un tamaño de bloque de datos grande, ODBC .NET Data Provider emitirá esta excepción:
	`Object reference not set to an instance of an object.`
	`at System.Runtime.InteropServices.Marshal.Copy(Int32 source, Char[] destination, Int32 startIndex, Int32 length)`
	`at System.Data.Odbc.OdbcDataReader.GetChars(Int32 i, Int64 dataIndex, Char[] buffer, Int32 bufferIndex, Int32 length)`
	`at OleRestrict.TestGetCharsAndBufferSize(IDbConnection con)`
		Todos

agrupación de conexiones	ODBC .NET Data Provider no controla la agrupación de conexiones. La agrupación de conexiones es gestionada por el Gestor de controladores ODBC. Para obtener más información sobre la agrupación de conexiones, consulte el manual de consulta del programador de ODBC, que se encuentra en la biblioteca de MSDN situada en http://msdn.microsoft.com/library.	Todos
DataColumnMapping	El nombre de la columna fuente debe coincidir exactamente con el nombre utilizado en las tablas de catálogo del sistema, que por omisión se escribe en mayúsculas.	Todos

Columnas decimales	No se pueden utilizar marcadores de parámetros para las columnas decimales.
	
	Generalmente utilizará `OdbcType.Decimal` para un `OdbcParameter` si el `SQLType` de destino es una columna `Decimal`; sin embargo, cuando ODBC .NET Data Provider lee el `OdbcType.Decimal`, vincula el parámetro utilizando el tipo C de `SQL_C_WCHAR` y `SQL_VARCHAR` como `SQLType`, lo cual no es válido.
	
	Por ejemplo:
	`[C#]`
	`cmd.CommandText = "SELECT dec_col FROM MYTABLE WHERE dec_col > ? ";`
	`OdbcParameter p1 = cmd.CreateParameter();`
	`p1.DbType = DbType.Decimal;`
	`p1.Value = 10.0;`
	`cmd.Parameters.Add(p1);`
	`IDataReader rdr = cmd.ExecuteReader();`
	
	Obtendrá una excepción:
	`ERROR [07006] [IBM][CLI Driver][SQLDS/VM] SQL0301N The value of input host variable or parameter number "" cannot be used because of its data type. SQLSTATE=07006`
	
	Corrección:
	
	En lugar de utilizar valores de `OdbcParameter`, utilice literales exclusivamente.
		DB2 para VM/VSE

Tabla 31. Restricciones de IBM ODBC .NET Data Provider (continuación)

Clase o función	Descripción de la restricción	Servidores DB2 afectados

Información de claves	El nombre de esquema utilizado para calificar el nombre de tabla (por ejemplo, `MYSCHEMA.MYTABLE`) debe coincidir con el ID de usuario de la conexión. ODBC .NET Data Provider no puede recuperar ninguna información de claves en la que el esquema especificado sea diferente del ID de usuario de la conexión.
	
	Por ejemplo:
	`CREATE TABLE USERID2.TABLE1(c1 INT NOT NULL PRIMARY KEY);`
	
	`[C#]`
	`// Conectar como usuario bob`
	`odbcCon = new OdbcConnection("DSN=sample;UID=bob;PWD=mypassword");`
	
	`OdbcCommand cmd = odbcCon.CreateCommand();`
	
	`// Seleccionar de la tabla con esquema USERID2`
	`cmd.CommandText="SELECT * FROM USERID2.TABLE1";`
	
	`// Error - no se recupera información de claves`
	`da.FillSchema(ds, SchemaType.Source);`
	
	`// Error - SchemaTable no tiene ninguna clave primaria`
	`cmd.ExecuteReader(CommandBehavior.KeyInfo)`
	
	`// Emite una excepción debido a la falta de claves primarias`
	`cbuilder.GetUpdateCommand();`
		Todos

Información de claves	ODBC .NET Data Provider no puede obtener información de claves al abrir un `IDataReader` al mismo tiempo. Cuando ODBC .NET Data Provider abre un `IDataReader`, se abre un cursor en el servidor. Si se solicita información de claves, se invocará `SQLPrimaryKeys()` o `SQLStatistic()` para obtener la información de claves, pero estas funciones de esquema abrirán otro cursor. Debido a que DB2 para VM/VSE no da soporte a la retención de cursor, se cierra el primer cursor. Como consecuencia, las llamadas de `IDataReader.Read()` al `IDataReader` producen la excepción siguiente:
	`System.Data.Odbc.OdbcException: ERROR [HY010] [IBM][CLI Driver]`
	`CLI0125E Function sequence error. SQLSTATE=HY010`
	
	Corrección:
	
	Primero deberá recuperar la información de claves y luego recuperar los datos.
	
	Por ejemplo:
	`[C#]`
	`OdbcCommand cmd = odbcCon.CreateCommand();`
	`OdbcDataAdapter da = new OdbcDataAdapter(cmd);`
	
	`cmd.CommandText = "SELECT * FROM MYTABLE";`
	
	`// Utilizar FillSchema para recuperar solo la información de esquema`
	`da.FillSchema(ds, SchemaType.Source);`
	`// Utilizar FillSchema para recuperar solo la información de esquema`
	`da.Fill(ds);`
		DB2 para VM/VSE

Información de claves	Los nombres utilizados en sus sentencias de SQL para especificar objetos de base de datos deben coincidir exactamente con los nombres utilizados para esos objetos en las tablas de catálogo del sistema. Por omisión, los nombres de los objetos de base de datos se guardan en mayúsculas en las tablas de catálogo del sistema. Generalmente, pues, deberá utilizar letras mayúsculas. ODBC .NET Data Provider examina sentencias de SQL para recuperar nombres de objetos de base de datos y los pasa a funciones de esquema, tales como `SQLPrimaryKeys` y `SQLStatistics`, que emiten consultas para estos objetos en las tablas de catálogo del sistema. Los nombres utilizados para especificar los objetos de base de datos deben coincidir exactamente con los nombres correspondientes que están almacenados en las tablas de catálogo del sistema; de lo contrario se devuelve un conjunto de resultados vacío.	DB2 para OS/390 DB2 para OS/400 DB2 para VM/VSE

Tabla 31. Restricciones de IBM ODBC .NET Data Provider (continuación)

Clase o función	Descripción de la restricción	Servidores DB2 afectados
Información de claves para sentencias de SQL de proceso por lotes distintas de SELECT	ODBC .NET Data Provider no puede recuperar ninguna información de claves para una sentencia de proceso por lotes que no comience por "SELECT".	DB2 para OS/390 DB2 para OS/400 DB2 para VM/VSE

Columnas LOB	ODBC .NET Data Provider no da soporte a los tipos de datos LOB. Como consecuencia, cada vez que el servidor DB2 devuelva un SQL_CLOB (-99), SQL_BLOB (-98) o SQL_DBCLOB (-350), ODBC .NET Data Provider emitirá la excepción siguiente:
	`"Unknown SQL type - -98"  (para una columna Blob)`
	`"Unknown SQL type - -99"  (para una columna Clob)`
	`"Unknown SQL type - -350"  (para una columna DbClob)`
	Cualquier método que acceda directa o indirectamente a columnas LOB fallará.
	Solución:
	Asigne el valor 1 a la palabra clave `LongDataCompat` de CLI/ODBC. Esto obligará al controlador CLI de DB2 a correlacionar los tipos de datos siguientes con tipos de datos que DBC .NET Data Provider reconoce, de esta manera:
	• SQL_CLOB con SQL_LONGVARCHAR
	• SQL_BLOB con SQL_LONGVARBINARY
	• SQL_DBCLOB con SQL_WLONGVARCHAR
	Para definir la palabra clave `LongDataCompat`, ejecute el mandato de DB2 siguiente desde una ventana de mandatos de DB2 en la máquina cliente:
	`db2 update cli cfg for section common using longdatacompat 1`
	Puede también definir esta palabra clave en su aplicación, utilizando la serie de conexión, de esta manera:
	`[C#]`
	`OdbcConnection con =`
	`  new OdbcConnection("DSN=SAMPLE;UID=uid;PWD=mypwd;LONGDATACOMPAT=1;");`
	Para obtener una lista de todas las palabras clave de CLI/ODBC, consulte el tema UID CLI/ODBC configuration keyword en el manual "DB2 Universal Database CLI Guide and Reference".
		Todos

`OdbcCommand.Cancel`	La ejecución de sentencias después de ejecutar `OdbcCommand.Cancel` puede producir la excepción siguiente: `"ERROR [24000] [Microsoft][ODBC Driver Manager] Invalid cursor state"`	Todos

Tabla 31. Restricciones de IBM ODBC .NET Data Provider (continuación)

Clase o función	Descripción de la restricción	Servidores DB2 afectados

OdbcCommandBuilder	La distinción entre letras mayúsculas y minúsculas es importante cuando se utiliza OdbcCommandBuilder para generar automáticamente sentencias UPDATE, DELETE e INSERT. Por omisión, DB2 almacena la información de esquema (tal como nombres de tabla y nombres de columna) en letras mayúsculas en las tablas de catálogo del sistema, a menos que se hayan creado explícitamente con discriminación de mayúsculas y minúsculas (mediante la adición de comillas a los nombres de objetos de base de datos en el momento de su creación). Las sentencias de SQL que defina deben utilizar nombres escritos exactamente tal como están almacenados en los catálogos (por omisión, los nombres se almacenan en mayúsculas).
	
	Por ejemplo, si ha creado una tabla utilizando la sentencia siguiente:
	`"db2 create table mytable (c1 int) "`
	
	DB2 almacenará el nombre de tabla ″mytable″ como ″MYTABLE″ en las tablas de catálogo del sistema.
	
	El ejemplo de código siguiente muestra la utilización apropiada de la clase OdbcCommandBuilder:
	`[C#]`
	`OdbcCommand cmd = odbcCon.CreateCommand();`
	`cmd.CommandText  = "SELECT * FROM MYTABLE";`
	`OdbcDataAdapter da = new OdbcDataAdapter(cmd);`
	`OdbcCommandBuilder cb = new OdbcCommandBuilder(da);`
	`OdbcCommand updateCmd = cb.GetUpdateCommand();`
	
	En este ejemplo, si no especifica el nombre de tabla en mayúsculas, obtendrá la excepción siguiente:
	`"La generación de SQL dinámico para UpdateCommand no está soportada para un SelectCommand que no devuelve ninguna información de columnas de clave."`
		Todos

OdbcCommandBuilder	Los mandatos generados por OdbcCommandBuilder son incorrectos cuando la sentencia SELECT contiene los tipos de datos de columna siguientes:
	`REAL`
	`FLOAT o DOUBLETIMESTAMP`
	
	Estos tipos de datos no se pueden utilizar en la cláusula WHERE de sentencias SELECT.
		DB2 para OS/390
		DB2 para OS/400
		DB2 para VM/VSE

OdbcCommandBuilder. DeriveParameters	El método DeriveParameters() se correlaciona con SQLProcedureColumns y utiliza la propiedad CommandText para el nombre del procedimiento almacenado. Debido a que CommandText no contiene el nombre del procedimiento almacenado (utilizando la sintaxis completa de la llamada de ODBC), SQLProcedureColumns se invoca con el nombre de procedimiento identificado de acuerdo con la sintaxis de la llamada de ODBC. Por ejemplo: `"{ CALL myProc(?) }"` Esto producirá un conjunto de resultados vacío, cuando no se encuentren columnas para el procedimiento.	Todos
OdbcCommandBuilder. DeriveParameters	Para utilizar DeriveParameters(), especifique el nombre de procedimiento almacenado en CommandText (por ejemplo, `cmd.CommandText = "MYPROC"`). El nombre del procedimiento almacenado debe estar escrito exactamente tal como está almacenado en las tablas de catálogo del sistema. DeriveParameters() devolverá todos los parámetros para ese nombre de procedimiento que encuentre en las tablas de catálogo del sistema. Recuerde que debe restaurar la sintaxis completa de llamada de ODBC para CommandText antes de ejecutar la sentencia.	Todos
OdbcCommandBuilder. DeriveParameters	El parámetro ReturnValue no se devuelve para ODBC .NET Data Provider.	Todos
OdbcCommandBuilder. DeriveParameters	DeriveParameters() no da soporte a nombres de procedimiento almacenado totalmente calificados. Por ejemplo, la invocación de DeriveParameters() para `CommandText = "MYSCHEMA.MYPROC"` fallará. En este caso, no se devuelve ningún parámetro.	Todos

Tabla 31. Restricciones de IBM ODBC .NET Data Provider (continuación)

Clase o función	Descripción de la restricción	Servidores DB2 afectados
OdbcCommandBuilder. DeriveParameters	`DeriveParameters()` no es efectivo para procedimientos almacenados sobrecargados. `SQLProcedureColumns` devolverá todos los parámetros para todas las versiones del procedimiento almacenado.	Todos
`OdbcConnection. ChangeDatabase`	El método `OdbcConnection.ChangeDatabase()` no está soportado.	Todos
`OdbcConnection. ConnectionString`	• La palabra clave `Server` no se tiene en cuenta. • La palabra clave `Connect Timeout` no se tiene en cuenta. La CLI de DB2 no da soporte a los tiempos de espera de conexión, por lo que el definir esta propiedad no afectará al controlador. • Las palabras clave de agrupación de conexiones no se tiene en cuenta. Específicamente, esto afecta a las palabras clave siguientes: `Pooling`, `Min Pool Size`, `Max Pool Size`, `Connection Lifetime` y `Connection Reset`.	Todos
OdbcDataReader. GetSchemaTable	ODBC .NET Data Provider no puede recuperar información descriptiva ampliada procedente de servidores que no devuelven esa clase de información. Por tanto, si se conecta a un servidor que no da soporte a información descriptiva ampliada (los servidores afectados), las columnas siguientes de la tabla de metadatos devuelta por `IDataReader.GetSchemaTable()` no son válidas: • `IsReadOnly` • `IsUnique` • `IsAutoIncrement` • `BaseSchemaName` • `BaseCatalogName`	DB2 para OS/390, versión 7 o inferior DB2 para OS/400 DB2 para VM/VSE

Procedimientos almacenados	Para llamar a un procedimiento almacenado debe especificar la sintaxis completa de llamada de ODBC.
	
	Por ejemplo, para llamar al procedimiento almacenado `MYPROC` que hace uso de un VARCHAR(10) como parámetro de entrada:
	`[C#]`
	`OdbcCommand cmd = odbcCon.CreateCommand();`
	`cmd.CommandType = CommandType.Text;`
	`cmd.CommandText = "{ CALL MYPROC(?) }"`
	`OdbcParameter p1 = cmd.CreateParameter();`
	`p1.Value = "Joe";`
	`p1.OdbcType = OdbcType.NVarChar;`
	`cmd.Parameters.Add(p1);`
	`cmd.ExecuteNonQuery();`
	
	Nota: Observe que debe utilizar la sintaxis completa de llamada de ODBC aunque esté utilizando `CommandType.StoredProcedure`. Esto está documentado en MSDN, en el tema sobre la propiedad OdbcCommand.CommandText.
		Todos

Procedimientos almacenados: sin nombres de columna para conjuntos de resultados	El servidor DB2 para OS/390 versión 6.1 no devuelve nombres de columna para conjuntos de resultados devueltos desde un procedimiento almacenado. ODBC .NET Data Provider correlaciona estas columnas sin nombre con su número de posición ordinal (por ejemplo, ″1″, ″2″ ″3″). Esto es diferente de la correlación documentada en MSDN: `"Columna1", "Columna2", "Columna3"`.	DB2 para OS/390 versión 6.1
Promoción de índice exclusivo a clave primaria	ODBC .NET Data Provider promociona los índices exclusivos anulables a claves primarias. Esto es contrario a la documentación de MSDN, que establece que los índices exclusivos anulables no se deben promocionar a claves primarias.	Todos

Parte 4. Java

Capítulo 14. Introducción al soporte de aplicaciones Java

DB2® Universal Database proporciona soporte de controlador para aplicaciones cliente y applets escritos Java™ utilizando JDBC, así como para SQL incorporado para Java (SQLJ).

JDBC es una interfaz de programación de aplicaciones (API) que las aplicaciones Java utilizan para acceder a bases de datos relacionales. El soporte de DB2 Universal Database™ para JDBC le permite escribir aplicaciones Java para acceder a datos DB2 locales o datos relacionales remotos situados en un servidor que da soporte a DRDA®.

SQLJ proporciona soporte para SQL estático incorporado en aplicaciones Java. SQLJ fue desarrollado inicialmente por IBM®, Oracle® y Tandem para complementar el modelo de SQL dinámico de JDBC con un modelo de SQL estático.

En general, las aplicaciones Java utilizan JDBC para el SQL dinámico y SQLJ para el SQL estático. Sin embargo, debido a que SQLJ puede interaccionar con JDBC, un programa de aplicación puede utilizar JDBC y SQLJ dentro de la misma unidad de trabajo.

El presente tema trata sobre el entorno de desarrollo de aplicaciones Java proporcionado por DB2 Universal Database.

De acuerdo con la especificación JDBC, existen cuatro tipos de arquitecturas de controlador JDBC:

Tipo 1
Son controladores que implementan la API de JDBC como una correlación con otra API de acceso a datos, tal como Open Database Connectivity (ODBC). Los controladores de este tipo generalmente dependen de una biblioteca nativa, lo cual limita su portabilidad. El controlador Bridge de JDBC-ODBC es un ejemplo de controlador de tipo 1.

Tipo 2
Son controladores que están escritos parcialmente en el lenguaje de programación Java y parcialmente en código nativo. Estos controladores utilizan una biblioteca cliente nativa que es específica de la fuente de datos a la que se conectan. Debido al código nativo, la portabilidad de estos controladores es limitada.

Tipo 3
Son controladores que utilizan un cliente Java puro y se comunican con un servidor utilizando un protocolo que es independiente de la base de datos. A su vez, el cliente transmite las peticiones del cliente a la fuente de datos.

Tipo 4
Estos controladores son Java puro e implementan el protocolo de red de una fuente de datos determinada. El cliente se conecta directamente con la fuente de datos.

DB2 Versión 8 soporta un controlador de tipo 2 y un controlador que combina las implementaciones JDBC de tipos 2 y 4. DB2 Versión 8 también soporta un controlador de tipo 3, pero el uso de este controlador está desaconsejado. Los

controladores JDBC existentes en releases anteriores de DB2 UDB para Linux, UNIX® y Windows® estaban basados en la CLI (Interfaz de nivel de llamada) de DB2. Los controladores de tipo 2 y tipo 3 de DB2 Versión 8 siguen utilizando la interfaz CLI de DB2 para comunicarse con los servidores DB2 UDB. DB2 Versión 8 añade un nuevo Controlador JDBC universal de DB2 que está escrito totalmente en Java. Los controladores que están soportados en DB2 Versión 8 son:

Controlador JDBC de DB2 de Tipo 2 para Linux, UNIX y Windows (controlador JDBC de DB2 de tipo 2) (uso no recomendado a partir de DB2 V8.2):

El controlador JDBC de DB2 de tipo 2 permite que las aplicaciones Java realicen llamadas a DB2 a través de JDBC. Las llamadas al controlador JDBC de DB2 de tipo 2 se convierten a métodos nativos Java. Las aplicaciones Java que hacen uso de este controlador se deben ejecutar en un cliente DB2, a través del cual las peticiones JDBC circulan hacia el servidor DB2. DB2 Connect™ Versión 8 debe estar instalado para que el controlador de aplicaciones JDBC de DB2 pueda acceder a fuentes de datos de DB2 UDB para iSeries™ o fuentes de datos de los entornos DB2 para OS/390 o z/OS.

El controlador JDBC de DB2 de tipo 2 soporta estas funciones de JDBC y SQLJ:
- La mayoría de los métodos que se describen en la especificación de JDBC 1.2 y algunos de los métodos que se describen en la especificación de JDBC 2.0. Consulte el tema Comparación del soporte de controlador para las API de JDBC.
- Sentencias de SQLJ que ejecutan funciones equivalentes para todos los métodos JDBC
- Agrupación de conexiones
- Transacciones distribuidas
- Funciones definidas por el usuario y procedimientos almacenados de Java

El Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows no estará soportado en futuros releases de DB2. Debe pues considerar la posibilidad de migrar hacia el Controlador JDBC universal de DB2.

Controlador JDBC de DB2 de tipo 3 para Linux, UNIX y Windows (uso no recomendado a partir de DB2 V8.1):

El controlador JDBC de DB2 de tipo 3, también conocido como applet o controlador de red, consta de un cliente JDBC y un servidor JDBC. El controlador de applet JDBC de DB2 puede ser cargado por un navegador Web junto con el applet, o se puede utilizar en aplicaciones Java autónomas. Cuando el applet solicita una conexión a un servidor de bases de datos DB2, el cliente abre un socket TCP/IP para el servidor de applet JDBC de DB2 en la máquina donde se ejecuta el servidor Web. Una vez establecida una conexión, el controlador de applet envía cada una de las peticiones subsiguientes de acceso a la base de datos desde el applet al servidor JDBC a través de la conexión TCP/IP. Luego el servidor JDBC realiza las llamadas correspondientes a DB2 para realizar la tarea. Una vez finalizado el proceso, el servidor JDBC envía los resultados al cliente JDBC a través de la conexión. El proceso del servidor JDBC es db2jd.

El Controlador JDBC de DB2 de tipo 3 para Linux, UNIX y Windows no estará soportado en futuros releases de DB2. Debe pues considerar la posibilidad de migrar hacia el Controlador JDBC universal de DB2.

Controlador JDBC universal de DB2 (tipo 2 y tipo 4):

El Controlador JDBC universal de DB2 es un controlador que incluye funciones de los tipos 2 y 4 de JDBC, así como soporte de SQLJ. Cuando una aplicación carga el Controlador JDBC universal de DB2, se carga una instancia de controlador para las implementaciones de tipo 2 y tipo 4. La aplicación puede establecer conexiones de tipo 2 y tipo 4 utilizando esta instancia de controlador. Las conexiones de 2 y tipo 4 se pueden establecer simultáneamente. La función del Controlador JDBC universal de DB2 de tipo 2 se denomina *Conectividad de tipo 2 del controlador JDBC universal de DB2.* La función del Controlador JDBC universal de DB2 de tipo 4 se denomina *Conectividad de tipo 4 del controlador JDBC universal de DB2.*

El Controlador JDBC universal de DB2 es un controlador completamente nuevo, y no una continuación de cualquiera de los demás controladores JDBC de DB2. Por tanto, son previsibles algunas diferencias de comportamiento entre este controlador y otros controladores.

El Controlador JDBC universal de DB2 soporta estas funciones de JDBC y SQLJ:
- La mayoría de los métodos que se describen en las especificaciones de JDBC 1.2 y JDBC 2.0, y algunos de los métodos que se describen en la especificación de JDBC 3.0. Consulte el tema Comparación del soporte de controlador para las API de JDBC.
- Sentencias de SQLJ que ejecutan funciones equivalentes para todos los métodos JDBC.
- Conexiones que están habilitadas para la agrupación de conexiones. WebSphere Application Server u otro servidor de aplicaciones realiza la agrupación de conexiones.
- Implementación de funciones definidas por el usuario y procedimientos almacenados de Java (solamente para Conectividad de tipo 2 universal).
- Transacciones globales que se ejecutan bajo WebSphere® Application Server Versión 5.0 y versiones superiores.
- Soporte para la gestión de transacciones distribuidas. Este soporte implementa las especificaciones Java 2 Platform, Enterprise Edition (J2EE), Java Transaction Service (JTS) y Java Transaction API (JTA), las cuales cumplen el estándar de X/Open para transacciones globales (*Distributed Transaction Processing: The XA Specification*, que se puede consultar en www.opengroup.org) .

Información relacionada:
- “Diferencias de JDBC entre el Controlador JDBC de DB2 Universal y otros controladores JDBC de DB2” en la página 459
- “Diferencias respecto a SQLJ entre el Controlador JDBC universal de DB2 y otros controladores JDBC de DB2” en la página 466
- “Comparación del soporte de controlador para las API de JDBC” en la página 407

Capítulo 15. Programación de aplicaciones JDBC

Las secciones siguientes contienen información sobre la escritura de aplicaciones JDBC.

Conceptos básicos de la programación de aplicaciones JDBC

Los temas siguientes contienen información básica sobre la escritura de aplicaciones JDBC.

Pasos básicos para escribir una aplicación JDBC

La escritura de una aplicación JDBC es muy similar a la escritura de una aplicación SQL en cualquier otro lenguaje: en general, es necesario seguir los pasos siguientes:

- Acceder a los paquetes Java™ donde están contenidos los métodos de JDBC.
- Declarar variables para enviar datos a tablas DB2® o para recuperarlos de ellas.
- Conectar con una fuente de datos.
- Ejecutar sentencias de SQL.
- Manejar los errores y avisos de SQL.
- Desconectar de la fuente de datos.

Aunque las tareas que necesita realizar son similares a las que se ejecutan en otros lenguajes, la forma de ejecutarlas es algo diferente.

La Figura 4 en la página 288 es un programa sencillo que muestra la ejecución de cada tarea. Este programa se ejecuta en el Controlador JDBC universal de DB2.

```
import java.sql.*;                                                          1

public class EzJava
{
  public static void main(String[] args)
  {
    String urlPrefix = "jdbc:db2:";
    String url;
    String empNo;                                                           2
    Connection con;
    Statement stmt;
    ResultSet rs;

    System.out.println ("**** Especificar clase EzJava");

    // Compruebe que el primer argumento tiene el formato correcto
    // para la porción del URL que sigue a jdbc:db2:, tal como se
    // describe en el tema Conectar con una fuente de datos utilizando
    // la interfaz DriverManager con el Controlador JDBC de DB2
    // Universal.
    // Por ejemplo, para la conectividad del Controlador Universal
    // de tipo 2, args[0] podría ser MVS1DB2M.
    // Para la conectividad del Controlador Universal de tipo 4,
    // args[0] podría ser //stlmvs1:10110/MVS1DB2M.
    if (args.length==0)
    {
      System.err.println ("Valor no válido. Primer argumento añadido a "+
       "jdbc:db2: debe especificar un URL válido.");
      System.exit(1);
    }
    url = urlPrefix + args[0];

    try
    {
      // Cargar el Controlador JDBC universal de DB2
      Class.forName("com.ibm.db2.jcc.DB2Driver");                           3a
      System.out.println("**** Controlador JDBC cargado");

      // Crear la conexión utilizando el Controlador JDBC de DB2 Universal
      con = DriverManager.getConnection (url);                              3b
      // Confirmar los cambios manualmente
      con.setAutoCommit(false);
      System.out.println("**** Creada una conexión JDBC con la fuente de datos");

      // Crear el objeto Statement
      stmt = con.createStatement();                                         4a
      System.out.println("**** Creado el objeto Statement de JDBC");

      // Ejecutar una consulta y generar instancia del conjunto de resultados
      rs = stmt.executeQuery("SELECT EMPNO FROM EMPLOYEE");                 4b
      System.out.println("**** Creado el objeto ResultSet de JDBC");

      // Imprimir todos los números de empleado en el dispositivo de salida estándar
      while (rs.next()) {
        empNo = rs.getString(1);
        System.out.println("Número de empleado = " + empNo);
      }
      System.out.println("**** Buscadas todas las filas del conjunto de resultados de JDBC");
```

Figura 4. Aplicación JDBC sencilla (Parte 1 de 2)

```
      // Cerrar el conjunto de resultados
      rs.close();
      System.out.println("**** Cerrado el conjunto de resultados de JDBC");

      // Cerrar el objeto Statement
stmt.close();
      System.out.println("**** Cerrado el objeto Statement de JDBC");

      // La conexión debe estar en un límite de unidad de trabajo para permitir el cierre
      con.commit();
      System.out.println ( "**** Transacción confirmada" );

      // Cierre la conexión
      con.close();                                                          6
      System.out.println("**** Desconectado de la fuente de datos");

      System.out.println("**** Salida de JDBC de la clase EzJava - sin errores");

    }

    catch (ClassNotFoundException e)
    {
      System.err.println("No se pudo cargar el controlador JDBC");
      System.out.println("Exception: " + e);
      e.printStackTrace();
    }

    catch(SQLException ex)                                                  5
    {
      System.err.println("Información sobre SQLException");
      while(ex!=null) {
        System.err.println ("Mensaje de error: " + ex.getMessage());
        System.err.println ("SQLSTATE: " + ex.getSQLState());
        System.err.println ("Código de error: " + ex.getErrorCode());
        ex.printStackTrace();
        ex = ex.getNextException(); // Para controladores que soportan
                                    // excepciones encadenadas
      }
    }
  }  // End main
}    // End EzJava
```

Figura 4. Aplicación JDBC sencilla (Parte 2 de 2)

Nota para la Figura 4 en la página 288:

1 Esta sentencia importa el paquete java.sql, el cual contiene la API básica de JDBC. Para obtener información sobre otros paquetes Java a los que puede necesitar acceder, consulte el tema Acceso a paquetes Java para el soporte de JDBC.

2 La variable empNo de tipo `String` realiza la función de una variable del lenguaje principal. Es decir, se utiliza para contener datos obtenidos en una consulta de SQL. Consulte el tema Declarar variables en aplicaciones JDBC para obtener más información.

3a y **3b** Estos dos conjuntos de sentencias muestran cómo conectar con una fuente de datos utilizando una de dos interfaces disponibles. Consulte el tema Conectar a una fuente de datos utilizando JDBC para obtener más detalles.

4a y **4b** Estos dos conjuntos de sentencias muestran cómo ejecutar una operación SELECT en JDBC. Para obtener información sobre cómo ejecutar otras operaciones de SQL, consulte el tema Ejecutar SQL en una aplicación JDBC.

5 Este bloque `try/catch` muestra el uso de la clase `SQLException` para el manejo de errores de SQL. Para obtener más información sobre el manejo de errores de SQL, consulte el tema Manejar una excepción de SQL cuando se utiliza el Controlador JDBC de DB2 Universal. Para obtener información sobre el manejo de avisos de SQL, consulte el tema Manejar avisos de SQL en una aplicación JDBC.

6 Esta sentencia desconecta la aplicación respecto de la fuente de datos. Consulte el tema Cerrar la conexión con la fuente de datos.

Conceptos relacionados:

- "Paquetes Java para el soporte JDBC" en la página 290
- "Variables en aplicaciones JDBC" en la página 290
- "Interfaces JDBC para ejecutar SQL" en la página 300
- "Conexión de las aplicaciones JDBC a una fuente de datos" en la página 291

Tareas relacionadas:

- "Manejo de una excepción de SQL cuando se utiliza el Controlador JDBC de DB2 Universal" en la página 308
- "Manejo de un aviso de SQL cuando se utiliza el Controlador JDBC de DB2 Universal" en la página 312

Paquetes Java para el soporte JDBC

Para invocar métodos de JDBC necesita poder acceder a los paquetes Java™ donde residen esos métodos. Puede hacerlo importando los paquetes o clases específicas, o bien utilizando los nombres de clase totalmente calificados. Puede necesitar los paquetes o clases siguientes para su programa de JDBC:

java.sql
Contiene la API básica de JDBC.

javax.naming
Contiene clases e interfaces para JNDI (Java Naming and Directory Interface), que a menudo se utiliza para implementar una fuente de datos (`DataSource`).

javax.sql
Contiene extensiones estándar de JDBC 2.0.

javax.transaction
Contiene soporte JDBC para transacciones distribuidas del Controlador JDBC de DB2® de tipo 2 para Linux, UNIX® y Windows® (Controlador JDBC de DB2 de tipo 2).

com.ibm.db2.jcc
Contiene la implementación de JDBC específica de DB2 para el controlador JDBC de DB2 Universal .

COM.ibm.db2.jdbc
Contiene la implementación específica de DB2 para el controlador JDBC de DB2 de tipo 2.

Variables en aplicaciones JDBC

Al igual que en cualquier otra aplicación Java™, es necesario declarar variables cuando escribe aplicaciones JDBC. En las aplicaciones Java, esas variables se denominan identificadores Java. Algunos de esos identificadores tienen la misma función que las variables de lenguaje principal utilizadas en otros lenguajes de programación: contienen datos que se intercambian con tablas DB2®. El identificador `empNo` contenido en el programa de ejemplo del tema Pasos básicos para escribir una aplicación JDBC es un ejemplo de un identificador Java de tipo `String` que contiene datos obtenidos de una columna CHAR de una tabla DB2.

Los tipos de datos Java que seleccione pueden afectar al rendimiento del sistema, pues DB2 elige mejores vías de acceso cuando los tipos de datos de las variables Java se corresponden estrechamente con los tipos de datos DB2. Tipos de datos de Java, JDBC y SQL muestra las correlaciones recomendadas de tipos de datos de Java y JDBC con tipos de datos de SQL.

Conceptos relacionados:

- "Pasos básicos para escribir una aplicación JDBC" en la página 287

Información relacionada:

- "Tipos de datos de Java, JDBC y SQL" en la página 395

Conexión de las aplicaciones JDBC a una fuente de datos

Para poder ejecutar sentencias de SQL en un programa SQL cualquiera, debe conectarse a un servidor de bases de datos. En JDBC, un servidor de bases de datos se denomina *fuente de datos*.

La Figura 5 muestra cómo una aplicación Java™ se conecta a una fuente de datos para un controlador de tipo 2 o Conectividad de tipo 2 del controlador JDBC universal de DB2.

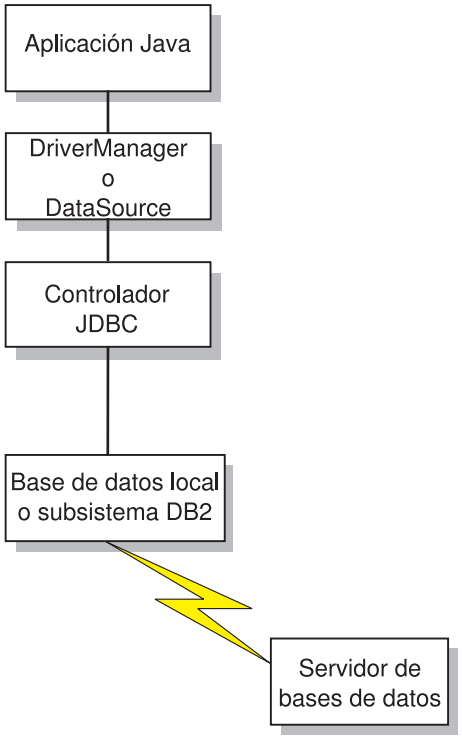
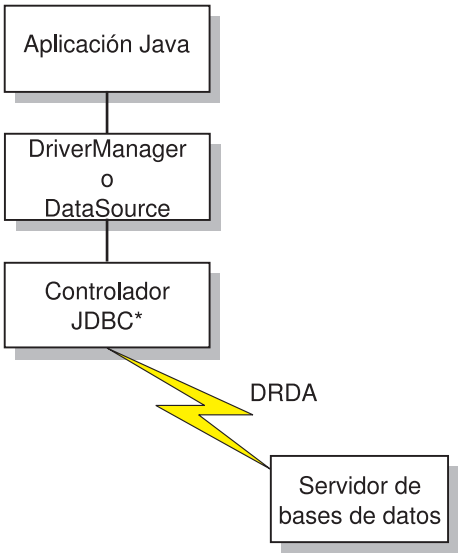


Figura 5. Flujo de una aplicación Java para un controlador de tipo 2 o Conectividad de tipo 2 del controlador JDBC universal de DB2

La Figura 6 en la página 292 muestra cómo una aplicación Java se conecta a una fuente de datos para Conectividad de tipo 4 del controlador JDBC universal de DB2.



*Código de bytes Java ejecutado bajo JVM

Figura 6. Flujo de aplicación Java para Conectividad de tipo 4 del controlador JDBC universal de DB2

La forma en que el usuario se conecta a la fuente de datos depende de la versión de JDBC que esté utilizando. Se puede establecer una conexión mediante la interfaz `DriverManager` para todos los niveles de JDBC. Se puede establecer una conexión mediante la interfaz `DataSource` para JDBC 2.0 y versiones posteriores.

Conceptos relacionados:

- "Conexión de aplicaciones DB2 a una fuente de datos utilizando la interfaz DriverManager con el Controlador JDBC de DB2 de tipo 2" en la página 292

Tareas relacionadas:

- "Conexión a una fuente de datos utilizando la interfaz DataSource" en la página 297
- "Conexión a una fuente de datos utilizando la interfaz DriverManager con el Controlador JDBC de DB2 Universal" en la página 294

Conexión de aplicaciones DB2 a una fuente de datos utilizando la interfaz DriverManager con el Controlador JDBC de DB2 de tipo 2

Una aplicación JDBC puede establecer una conexión con una fuente de datos utilizando la interfaz `DriverManager` de JDBC, la cual forma parte del paquete `java.sql`.

Primero la aplicación Java™ carga el controlador JDBC invocando el método `Class.forName`. Después de cargar el controlador, la aplicación conecta con un servidor de bases de datos invocando el método `DriverManager.getConnection`.

Para el Controlador JDBC de DB2® de tipo 2 para Linux, UNIX® y Windows® (Controlador JDBC de DB2 de tipo 2), el controlador se carga invocando el método `Class.forName` con el argumento siguiente:

```
COM.ibm.db2.jdbc.app.DB2Driver
```

El código siguiente muestra la carga del Controlador JDBC de DB2 de tipo 2:

```
try {
  // Cargar el Controlador JDBC de DB2 de tipo 2 con DriverManager
  Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");
} catch (ClassNotFoundException e) {
    e.printStackTrace();
}
```

El bloque `catch` se utiliza para imprimir un error si se no se encuentra el controlador.

Después de cargar el controlador, se conecta a la fuente de datos invocando el método `DriverManager.getConnection`. Puede utilizar uno de los formatos siguientes de `getConnection`:

```
getConnection(String url);
getConnection(String url, usuario , contraseña);
getConnection(String url, java.util.Properties info);
```

El argumento *url* representa una fuente de datos.

Para el Controlador JDBC de DB2 de tipo 2, especifique un URL de la forma siguiente:

Sintaxis de un URL para el Controlador JDBC de DB2 de tipo 2:

Los elementos del URL tienen los significados siguientes:

jdbc:db2:
jdbc:db2: indica que la conexión es con un servidor DB2 UDB.

basedatos
Es un alias de base de datos. El alias hace referencia a la entrada del catálogo de bases de datos DB2 que existe en el cliente DB2.

El argumento *info* es un objeto de tipo `java.util.Properties` que contiene un conjunto de propiedades de controlador correspondientes a la conexión. El argumento *infor* se especifica como alternativa a especificar las series *propiedad=valor* en el URL.

Especificación de un ID de usuario y contraseña para una conexión: existen varias formas de especificar un ID de usuario y una contraseña para una conexión:

- Utilice el formato del método `getConnection` en el que se especifica el *usuario* y *contraseña*.
- Utilice el formato del método `getConnection` en el que se especifica *info* después de definir las propiedades correspondientes al usuario y contraseña en un objeto `java.util.Properties`.

Ejemplo: definir el ID de usuario y contraseña en los parámetros de usuario y contraseña:

```
String url = "jdbc:db2:toronto";
                        // Definir URL para fuente de datos
String user = "db2adm";
String password = "db2adm";
Connection con = DriverManager.getConnection(url, user, password);
                            // Crear conexión
```

Ejemplo: definir el ID de usuario y contraseña en un objeto `java.util.Properties`:

```
Properties properties = new Properties(); // Crear objeto Properties
properties.put("user", "db2adm");  // Definir ID de usuario para conexión
properties.put("password", "db2adm");  // Definir contraseña para conexión
String url = "jdbc:db2:toronto";
                                       // Definir URL para fuente de datos
Connection con = DriverManager.getConnection(url, properties);
                                       // Crear conexión
```

Conceptos relacionados:

- "Seguridad cuando se utiliza el Controlador JDBC de DB2 de tipo 2" en la página 477

Conexión a una fuente de datos utilizando la interfaz DriverManager con el Controlador JDBC de DB2 Universal

Una aplicación JDBC puede establecer una conexión con una fuente de datos utilizando la interfaz `DriverManager` de JDBC, la cual forma parte del paquete `java.sql`.

Primero la aplicación Java™ carga el controlador JDBC invocando el método `Class.forName`. Después de cargar el controlador, la aplicación conecta con un servidor de bases de datos invocando el método `DriverManager.getConnection`.

Para el Controlador JDBC universal de DB2, el controlador se carga invocando el método `Class.forName` con este argumento:

```
com.ibm.db2.jcc.DB2Driver
```

Para mantener la compatibilidad con controladores JDBC anteriores, puede utilizar el argumento siguiente como alternativa:

```
COM.ibm.db2os390.sqlj.jdbc.DB2SQLJDriver
```

El código siguiente muestra la carga del Controlador JDBC universal de DB2:

```
try {
  // Cargar el controlador JDBC de DB2® Universal mediante DriverManager
  Class.forName("com.ibm.db2.jcc.DB2Driver");
} catch (ClassNotFoundException e) {
     e.printStackTrace();
}
```

El bloque `catch` se utiliza para imprimir un error si se no se encuentra el controlador.

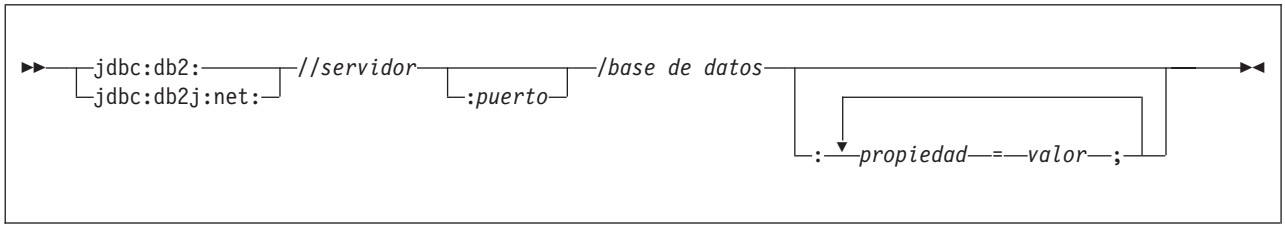
Después de cargar el controlador, se conecta a la fuente de datos invocando el método `DriverManager.getConnection`. Puede utilizar uno de los formatos siguientes de `getConnection`:

```
getConnection(String  url);
getConnection(String url, usuario, contraseña);
getConnection(String url, java.util.Properties info);
```

El argumento *url* representa una fuente de datos e indica el tipo de conectividad JDBC que se está utilizando.

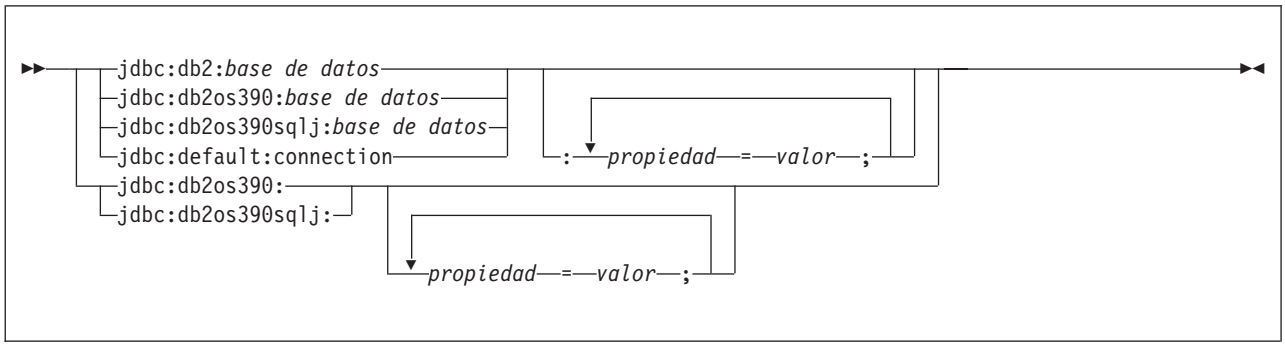
Para Conectividad de tipo 4 del controlador JDBC universal de DB2, especifique un URL de la forma siguiente:

Sintaxis para un URL de Conectividad de tipo 4 universal:



Para Conectividad de tipo 2 del controlador JDBC universal de DB2, especifique un URL en una de las formas siguientes:

Sintaxis para un URL de Conectividad de tipo 2 universal:



Los elementos del URL tienen los significados siguientes:

jdbc:db2: o jdbc:db2j:net:
El significado de la porción inicial del URL es el siguiente:

jdbc:db2:
Indica que la conexión es con un servidor perteneciente a la familia de productos DB2 UDB.

jdbc:db2j:net:
Indica que la conexión es con un servidor remoto IBM® Cloudscape™.

servidor
El nombre de dominio o dirección IP del servidor de bases de datos.

puerto
El número de puerto del servidor TCP/IP que está asignado al servidor de bases de datos. Es un valor entero comprendido entre 0 y 65535. El valor por omisión es 446.

basedatos
Nombre del servidor de bases de datos. Este nombre depende de si se utiliza Conectividad de tipo 4 universal o Conectividad de tipo 2 universal.

Para Conectividad de tipo 4 universal:

- Si la conexión es con un servidor DB2 para z/OS, el valor de *basedatos* es el nombre de ubicación de DB2 que se define durante la instalación. Todos los caracteres de este valor deben ser caracteres en mayúsculas. Puede determinar el nombre de ubicación ejecutando la sentencia de SQL siguiente en el servidor:

  ```
  SELECT CURRENT SERVER FROM SYSIBM.SYSDUMMY1;
  ```

- Si la conexión es con un servidor DB2 UDB para Linux, UNIX y Windows, el valor de *basedatos* es el nombre de base de datos que se define durante la instalación.
- Si la conexión es con un servidor IBM Cloudscape, el valor de *basedatos* es el nombre totalmente calificado del archivo donde reside la base de datos. Este nombre debe estar encerrado entre comillas dobles ("). Por ejemplo:

  ```
  "c:/basedatos/testdb"
  ```

Para Conectividad de tipo 2 universal:

- *basedatos* es el nombre de base de datos que se define durante la instalación, si el valor de la propiedad de conexión serverName es nulo. Si el valor de la propiedad serverName no es nulo, *basedatos* es un alias de base de datos.

propiedad=valor;
Es una propiedad de la conexión JDBC. Para conocer las definiciones de estas propiedades, consulte el tema Propiedades del Controlador JDBC de DB2 Universal.

El argumento *info* es un objeto de tipo `java.util.Properties` que contiene un conjunto de propiedades de controlador correspondientes a la conexión. El argumento *infor* se especifica como alternativa a especificar las series *propiedad=valor* en el URL. Consulte el tema Propiedades del Controlador JDBC de DB2 Universal para conocer las propiedades que puede especificar.

Especificación de un ID de usuario y contraseña para una conexión: existen varias formas de especificar un ID de usuario y una contraseña para una conexión:

- Utilice la modalidad del método `getConnection` por el que se especifica el *url* con cláusulas `propiedad=`*valor*;, e incluya las propiedades correspondientes al usuario y contraseña en el URL.
- Utilice la modalidad del método `getConnection` por el que se especifica el *usuario* y *contraseña*.
- Utilice la modalidad del método `getConnection` en la que se especifica *info*, después de definir las propiedades correspondientes al usuario y contraseña en un objeto `java.util.Properties`.

Ejemplo: definir el ID de usuario y contraseña en un URL:

```
String url = "jdbc:db2://sysmvs1.stl.ibm.com:5021/san_jose:" +
  "user=db2adm;password=db2adm;";
                        // Definir URL para fuente de datos
Connection con = DriverManager.getConnection(url);
                        // Crear conexión
```

Ejemplo: definir el ID de usuario y contraseña en los parámetros de usuario y contraseña:

```
String url = "jdbc:db2://sysmvs1.stl.ibm.com:5021/san_jose";
                       // Definir URL para fuente de datos
String user = "db2adm";
String password = "db2adm";
Connection con = DriverManager.getConnection(url, user, password);
                        // Crear conexión
```

Ejemplo: definir el ID de usuario y contraseña en un objeto `java.util.Properties`:

```
Properties properties = new Properties(); // Crear objeto Properties
properties.put("user", "db2adm");       // Definir ID de usuario para conexión
properties.put("password", "db2adm");  // Definir contraseña para conexión
String url = "jdbc:db2://sysmvs1.stl.ibm.com:5021/san_jose";
```

```
                                        // Definir URL para fuente de datos
Connection con = DriverManager.getConnection(url, properties);
                                        // Crear conexión
```

Conceptos relacionados:

- "Seguridad cuando se utiliza el Controlador JDBC de DB2 Universal" en la página 478

Información relacionada:

- "Propiedades del Controlador JDBC de DB2 Universal" en la página 400

Conexión a una fuente de datos utilizando la interfaz DataSource

El uso de DriverManager para conectar con una fuente de datos reduce la portabilidad, pues la aplicación debe identificar un nombre de clase y URL determinados para el controlador JDBC. El nombre de clase y el URL del controlador son específicos de un proveedor de JDBC, de la implementación del controlador y de la fuente de datos. Si es necesario que sus aplicaciones se puedan migrar de una fuente de datos a otra, debe utilizar la interfaz DataSource.

Cuando se conecta a una fuente de datos utilizando la interfaz DataSource, utiliza un objeto DataSource. Puede crear y utilizar el objeto DataSource en la misma aplicación, tal como lo hace en la interfaz DriverManager. La Figura 7 muestra un ejemplo para el Controlador JDBC universal de DB2:

Figura 7. Creación y utilización de un objeto DataSource en la misma aplicación

```
import java.sql.*;    // Base JDBC
import javax.sql.*;       // Las API de extensión estándar de JDBC 2.0
import com.ibm.db2.jcc.*; // Interfaces del controlador  1
                          // JDBC de DB2® Universal
 DB2SimpleDataSource db2ds=new DB2SimpleDataSource();     2
db2ds.setDatabaseName("db2loc1");                         3
                          // Asignar el nombre de ubicación
db2ds.setDescription("Our Sample Database");
                          // Descripción con fines de documentación
db2ds.setUser("john");
                          // Asignar el ID de usuario
db2ds.setPassword("db2");
                          // Asignar la contraseña
Connection con=db2ds.getConnection();                     4
                          // Crear un objeto Connection
```

1. Importa el paquete donde reside la implementación de la interfaz DataSource.
2. Crea un objeto DB2DataSource. DB2DataSource es una de las implementaciones DB2 de la interfaz DataSource. Consulte el tema Crear y desplegar objetos DataSource para obtener información sobre las implementaciones de DataSource para DB2.
3. Los métodos setDatabaseName, setDescription, setUser y setPassword asignan atributos al objeto DB2DataSource. Consulte el tema Propiedades del Controlador JDBC de DB2 Universal para obtener información sobre los atributos que puede definir para un objeto DB2DataSource cuando se utiliza el Controlador JDBC universal de DB2.
4. Establece una conexión con la fuente de datos representada por el objeto db2ds de DB2DataSource.

Sin embargo, una forma más flexible de utilizar un objeto DataSource consiste en que el administrador del sistema cree y gestione el objeto por separado, utilizando WebSphere® o alguna otra herramienta. El programa encargado de crear y

gestionar un objeto a DataSource también utiliza JNDI (Java™ Naming and Directory Interface) para asignar un nombre lógico al objeto DataSource. La aplicación JDBC que hace uso del objeto DataSource puede luego referirse al objeto utilizando su nombre lógico, y no necesita ninguna información sobre la fuente de datos subyacente. Además, el administrador del sistema puede modificar los atributos de la fuente de datos, y el usuario no necesita cambiar su programa de aplicación.

Para aprender más sobre la utilización de WebSphere para desplegar objetos DataSource, vaya a este URL de la Web:

`http://www.ibm.com/software/webservers/appserv/`

Para aprender sobre el despliegue de objetos DataSource, consulte el tema Crear y desplegar objetos DataSource.

Puede utilizar la interfaz DataSource y la interfaz DriverManager en la misma aplicación, pero para lograr una portabilidad máxima es recomendable que utilice solo la interfaz DataSource para obtener conexiones.

El resto de este tema explica cómo crear una conexión utilizando un objeto DataSource, dando por supuesto que el administrador del sistema ya ha creado el objeto y le ha asignado un nombre lógico.

Para obtener una conexión utilizando un objeto DataSource, siga estos pasos:

1. Consulte al administrador del sistema para conocer el nombre lógico de la fuente de datos con la que desea conectar.
2. Cree un objeto Context para utilizarlo en el paso siguiente. La interfaz Context forma parte de JNDI (Java Naming and Directory Interface), no de JDBC.
3. En su programa de aplicación, utilice JNDI para obtener el objeto DataSource que está asociado al nombre lógico de la fuente de datos.
4. Utilice el método DataSource.getConnection para obtener la conexión.

   Puede utilizar uno de los formatos siguientes del método getConnection:

   ```
   getConnection();
   getConnection(String usuario, String contraseña);
   ```

   Utilice la segunda forma si necesita especificar un ID de usuario y una contraseña para la conexión que sean diferentes de los que se especificaron al desplegar el objeto DataSource.

La Figura 8 muestra un ejemplo del código necesario en su programa de aplicación para obtener una conexión utilizando un objeto DataSource, donde jdbc/sampledb es el nombre lógico de la fuente de datos con la que desea conectar. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

Figura 8. Obtención de una conexión utilizando un objeto DataSource

```
import java.sql.*;
import javax.naming.*;
import javax.sql.*;
...
Context ctx=new InitialContext();                          2
DataSource ds=(DataSource)ctx.lookup("jdbc/sampledb");     3
Connection con=ds.getConnection();                         4
```

Tareas relacionadas:

- "Creación y despliegue de objetos DataSource" en la página 338

Información relacionada:

- "Propiedades del Controlador JDBC de DB2 Universal" en la página 400

Establecimiento del nivel de aislamiento para una transacción de JDBC

Para establecer el nivel de aislamiento para una unidad de trabajo dentro de un programa JDBC, utilice el método `Connection.setTransactionIsolation(int nivel)`. La Tabla 32 muestra los valores de *nivel* que puede especificar en el método `Connection.setTransactionIsolation` y los valores equivalentes en DB2®.

Tabla 32. Niveles de aislamiento equivalentes de JDBC y DB2

Valor de JDBC	Nivel de aislamiento de DB2
TRANSACTION_SERIALIZABLE	Lectura repetible
TRANSACTION_REPEATABLE_READ	Estabilidad de lectura
TRANSACTION_READ_COMMITTED	Estabilidad del cursor
TRANSACTION_READ_UNCOMMITTED	Lectura no confirmada

Puede cambiar el nivel de aislamiento solo al comienzo de una transacción.

Conceptos relacionados:

- "Objetos de conexión JDBC" en la página 299

Objetos de conexión JDBC

Cuando se conecta a una fuente de datos mediante cualquiera de los dos métodos de conexión, crea un objeto `Connection`, que representa la conexión con la fuente de datos. Utilice este objeto `Connection` para realizar lo siguiente:

- Crear objetos `Statement`, `PreparedStatement`, y `CallableStatement` para ejecutar sentencias de SQL. Esto se trata en el tema Ejecutar SQL en una aplicación JDBC.
- Reunir información sobre la fuente de datos a la que está conectado. Este proceso se trata en el tema Uso de DatabaseMetaData para obtener información sobre una fuente de datos.
- Confirmar o retrotraer transacciones. Puede confirmar transacciones de forma manual o automática. Estas operaciones se tratan en el tema Confirmar o retrotraer una transacción JDBC.
- Cerrar la conexión con la fuente de datos. Esta operación se trata en el tema Cerrar una conexión con una fuente de datos JDBC.

Conceptos relacionados:

- "Interfaces JDBC para ejecutar SQL" en la página 300

Tareas relacionadas:

- "Cierre de una conexión con una fuente de datos JDBC" en la página 300
- "Confirmación o retrotracción de transacciones JDBC" en la página 300
- "Uso de DatabaseMetaData para obtener información sobre una fuente de datos" en la página 327

Confirmación o retrotracción de transacciones JDBC

En JDBC, para confirmar o retrotraer explícitamente transacciones, utilice los métodos `commit` o `rollback`. Por ejemplo:

```
Connection con;
...
con.commit();
```

Si la modalidad de confirmación automática está activa, DB2® realiza una operación de confirmación después de finalizar cada sentencia de SQL. Para determinar si la modalidad de confirmación automática está activa, invoque el método `Connection.getAutoCommit`. Para activar la modalidad de confirmación automática, invoque el método `Connection.setAutoCommit(true)`. Para desactivar la modalidad de confirmación automática, invoque el método `Connection.setAutoCommit(false)`.

Conceptos relacionados:

- "Puntos de rescate en aplicaciones JDBC" en la página 320

Tareas relacionadas:

- "Realización de actualizaciones por lotes en aplicaciones JDBC" en la página 330
- "Cierre de una conexión con una fuente de datos JDBC" en la página 300

Cierre de una conexión con una fuente de datos JDBC

Una vez finalizada una conexión con una fuente de datos, es *esencial* que cierre la conexión con la fuente de datos. De esta forma se liberan inmediatamente los recursos del objeto `Connection` de DB2® y JDBC. Para cerrar la conexión con la fuente de datos, utilice el método `close`. Por ejemplo:

```
Connection con;
...
     con.close();
```

Si la modalidad de confirmación automática no está activa, es necesario que la conexión se encuentre en los límites de una unidad de trabajo antes de cerrar la conexión.

Conceptos relacionados:

- "Conexión de las aplicaciones JDBC a una fuente de datos" en la página 291

Interfaces JDBC para ejecutar SQL

Puede ejecutar sentencias de SQL dentro de un programa SQL típico para insertar, actualizar, suprimir o recuperar datos de tablas, o invocar procedimientos almacenados. Para realizar las mismas funciones en un programa JDBC, debe invocar métodos que están definidos en las interfaces siguientes:

- La interfaz `Statement` soporta la ejecución de todas las sentencias de SQL. Las interfaces siguientes heredan métodos de la interfaz `Statement`:
  - La interfaz `PreparedStatement` soporta cualquier sentencia de SQL que contenga marcadores de parámetros de entrada. Los marcadores de parámetros representan variables de entrada. La interfaz `PreparedStatement` también se puede utilizar para sentencias de SQL sin marcadores de parámetros.

Con el Controlador JDBC universal de DB2, la interfaz `PreparedStatement` permite invocar procedimientos almacenados que tienen parámetros de entrada y ningún parámetro de salida, y que no devuelven ningún conjunto de resultados.

– La interfaz `CallableStatement` soporta la invocación de un procedimiento almacenado.

  La interfaz `CallableStatement` permite invocar procedimientos almacenados con parámetros de entrada, parámetros de salida, con ambas clases de parámetros o sin parámetros. Con el Controlador JDBC universal de DB2, puede también utilizar la interfaz `Statement` para invocar procedimientos almacenados, pero éstos no deben tener parámetros.

- La interfaz `ResultSet` proporciona acceso a los resultados generados por una consulta. La interfaz `ResultSet` tiene la misma finalidad que el cursor utilizado en las aplicaciones de SQL para otros lenguajes de programación.

Para obtener una lista completa del soporte de DB2® para interfaces JDBC, consulte el tema Comparación del soporte de controlador para las API de JDBC.

Tareas relacionadas:

- "Uso del método PreparedStatement.executeQuery para obtener datos de DB2" en la página 305
- "Uso del método PreparedStatement.executeUpdate para actualizar datos de tablas DB2" en la página 303
- "Utilización del método Statement.executeQuery para recuperar datos de tablas DB2" en la página 302
- "Utilización del método Statement.executeUpdate para crear y modificar objetos DB2" en la página 301

Información relacionada:

- "Comparación del soporte de controlador para las API de JDBC" en la página 407

Utilización del método Statement.executeUpdate para crear y modificar objetos DB2

Puede utilizar el método `Statement.executeUpdate` para realizar las acciones siguientes:

- Ejecutar sentencias de definición de datos, tales como CREATE, ALTER, DROP, GRANT y REVOKE
- Ejecutar sentencias INSERT, UPDATE y DELETE que no contienen marcadores de parámetros
- Con el Controlador JDBC universal de DB2, ejecutar la sentencia CALL para invocar procedimientos almacenados que carecen de parámetros y no devuelven conjuntos de resultados.

Para ejecutar esas sentencias de SQL, debe seguir estos pasos:

1. Invoque el método `Connection.createStatement` para crear un objeto `Statement`.
2. Invoque el método `Statement.executeUpdate` para ejecutar la operación de SQL.
3. Invoque el método `Statement.close` para cerrar el objeto `Statement`.

Por ejemplo, suponga que desea ejecutar esta sentencia de SQL:

```
UPDATE EMPLOYEE SET PHONENO='4657' WHERE EMPNO='000010'
```

El código siguiente crea el objeto Statement denominado stmt, ejecuta la sentencia UPDATE y devuelve en numUpd el número de filas que fueron actualizadas. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
Connection con;
Statement stmt;
int numUpd;
...
stmt = con.createStatement();     // Crear un objeto Statement   1
numUpd = stmt.executeUpdate(
  "UPDATE EMPLOYEE SET PHONENO='4657' WHERE EMPNO='000010'");    2
                                  // Ejecutar la actualización
stmt.close();                     // Cerrar el objeto Statement  3
```

Figura 9. Utilización de Statement.executeUpdate

Información relacionada:

- "Diferencias de JDBC entre el Controlador JDBC de DB2 Universal y otros controladores JDBC de DB2" en la página 459
- "Comparación del soporte de controlador para las API de JDBC" en la página 407

Utilización del método Statement.executeQuery para recuperar datos de tablas DB2

Para recuperar datos de una tabla utilizando una sentencia SELECT sin marcadores de parámetros, puede utilizar el método Statement.executeQuery. Este método devuelve una tabla de resultados en un objeto ResultSet. Una vez obtenida la tabla de resultados, debe utilizar métodos de ResultSet para desplazarse por la tabla de resultados y obtener los valores de cada columna de cada fila.

Con el Controlador JDBC universal de DB2, también puede utilizar el método Statement.executeQuery para obtener un conjunto de resultados de una llamada de procedimiento almacenado, si ese procedimiento almacenado devuelve un solo conjunto de resultados. Si el procedimiento almacenado devuelve varios conjuntos de resultados, debe utilizar el método Statement.execute. Consulte el tema Obtención de varios conjuntos de resultados de un procedimiento almacenado en una aplicación JDBC para obtener más información.

Este tema describe la modalidad más sencilla de ResultSet, que es un objeto ResultSet de solo lectura en el que el usuario solo puede desplazarse hacia delante, una fila cada vez. El Controlador JDBC universal de DB2 también da soporte a objetos ResultSet actualizables y desplazables. Esto se describe en el tema Especificación de las capacidades de actualización, desplazamiento y retención para conjuntos de resultados en aplicaciones JDBC.

Para recuperar filas de una tabla utilizando una sentencia SELECT sin marcadores de parámetros, siga estos pasos:

1. Invoque el método Connection.createStatement para crear un objeto Statement.
2. Invoque el método Statement.executeQuery para obtener la tabla de resultados de la sentencia SELECT en un objeto ResultSet.
3. En un bucle, posicione el cursor utilizando el método next y recupere datos de cada columna de la fila actual del objeto ResultSet utilizando métodos getXXX.

XXX representa un tipo de datos. Consulte el tema Comparación del soporte de controlador para las API de JDBC para obtener una lista de los métodos getXXX y setXXX soportados.

4. Invoque el método ResultSet.close para cerrar el objeto ResultSet.
5. Invoque el método Statement.close para cerrar el objeto Statement cuando termine de utilizar ese objeto.

Por ejemplo, el código siguiente muestra cómo recuperar todas las filas de la tabla de empleados (EMPLOYEE). Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
String empNo;
Connection con;
Statement stmt;
    ResultSet rs;
...
stmt = con.createStatement();     // Crear un objeto Statement   1
rs = stmt.executeQuery("SELECT EMPNO FROM EMPLOYEE");            2
                                  // Obtener tabla de resultados de la consulta
while (rs.next()) {               // Posicionar el cursor          3
 empNo = rs.getString(1);         // Obtener solo el valor de la primera columna
 System.out.println("Número de empleado = " + empNo);
                                  // Imprimir el valor de columna
}
rs.close();                       // Cerrar ResultSet    4
stmt.close();                     // Cerrar Statement    5
```

Figura 10. Utilización de Statement.executeQuery

Tareas relacionadas:

- "Recuperación de varios conjuntos de resultados de un procedimiento almacenado en una aplicación JDBC" en la página 323
- "Especificación de las capacidades de actualización, desplazamiento y retención para conjuntos de resultados en aplicaciones de JDBC" en la página 335

Información relacionada:

- "Comparación del soporte de controlador para las API de JDBC" en la página 407

Uso del método PreparedStatement.executeUpdate para actualizar datos de tablas DB2

El método Statement.executeUpdate es efectivo si actualiza tablas DB2® con valores constantes. Sin embargo, a menudo las actualizaciones comportan el paso a tablas DB2 de valores contenidos en variables. Para hacer esto, utilice el método PreparedStatement.executeUpdate.

Con el Controlador JDBC universal de DB2, puede también utilizar PreparedStatement.executeUpdate para invocar procedimientos almacenados que tienen parámetros de entrada y ningún parámetro de salida, y que no devuelven ningún conjunto de resultados.

Cuando ejecuta una sentencia de SQL muchas veces, puede obtener un mejor rendimiento creando la sentencia de SQL en forma de objeto PreparedStatement.

Por ejemplo, la siguiente sentencia UPDATE permite actualizar la tabla de empleados (EMPLOYEE) solamente para un único número de teléfono y número de empleado:

```
UPDATE EMPLOYEE SET PHONENO='4657' WHERE EMPNO='000010'
```

Suponga que desea generalizar la operación para poder actualizar la tabla de empleados para un conjunto cualquiera de números de teléfono y números de empleado. Para ello es necesario que sustituya los valores constantes del número de teléfono y número de empleado por variables:

```
UPDATE EMPLOYEE SET PHONENO=? WHERE EMPNO=?
```

Las variables de esta clase se denominan marcadores de parámetros. Para ejecutar una sentencia de SQL con marcadores de parámetros, debe seguir estos pasos:

1. Invoque el método `Connection.prepareStatement` para crear un objeto `PreparedStatement`.
2. Invoque métodos `PreparedStatement.setXXX` para pasar valores a las variables.
3. Invoque el método `PreparedStatement.executeUpdate` para actualizar la tabla con los valores variables.
4. Invoque el método `PreparedStatement.close` para cerrar el objeto `PreparedStatement` cuando termine de utilizar ese objeto.

El código siguiente ejecuta los pasos anteriores para actualizar el número de teléfono '4657' del empleado cuyo número de empleado es '000010'. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
Connection con;
PreparedStatement pstmt;
int numUpd;
...
pstmt = con.prepareStatement(
  "UPDATE EMPLOYEE SET PHONENO=? WHERE EMPNO=?");
                          // Crear un objeto PreparedStatement   1
pstmt.setString(1,"4657");     // Asignar valor al primer parámetro   2
pstmt.setString(2,"000010");   // Asignar valor al segundo parámetro de entrada
numUpd = pstmt.executeUpdate();   // Ejecutar la actualización            3
pstmt.close();                    // Cerrar el objeto PreparedStatement   4
```

Figura 11. Uso de PreparedStatement.executeUpdate para una sentencia de SQL con marcadores de parámetros

Puede también utilizar el método `PreparedStatement.executeUpdate` para sentencias que no tienen marcadores de parámetros. Los pasos para ejecutar un objeto `PreparedStatement` sin marcadores de parámetros son similares a los pasos para ejecutar el objeto `PreparedStatement` con marcadores de parámetros, excepto que se omite el paso 2. El ejemplo siguiente muestra estos pasos.

```
Connection con;
PreparedStatement pstmt;
int numUpd;
...
pstmt = con.prepareStatement(
  "UPDATE EMPLOYEE SET PHONENO='4657' WHERE EMPNO='000010'");
                                  // Crear un objeto PreparedStatement  1
numUpd = pstmt.executeUpdate();   // Ejecutar la actualización          3
pstmt.close();                    // Cerrar el objeto PreparedStatement 4
```

Figura 12. Uso de PreparedStatement.executeUpdate para una sentencia de SQL sin marcadores de parámetros

Información relacionada:

- "Diferencias de JDBC entre el Controlador JDBC de DB2 Universal y otros controladores JDBC de DB2" en la página 459
- "Comparación del soporte de controlador para las API de JDBC" en la página 407

Uso del método PreparedStatement.executeQuery para obtener datos de DB2

Para obtener datos de una tabla utilizando una sentencia SELECT con marcadores de parámetros, utilice el método `PreparedStatement.executeQuery`. Este método devuelve una tabla de resultados en un objeto `ResultSet`. Una vez obtenida la tabla de resultados, debe utilizar métodos de `ResultSet` para desplazarse por la tabla de resultados y obtener los valores de cada columna de cada fila.

Con el Controlador JDBC universal de DB2, también puede utilizar el método `PreparedStatement.executeQuery` para obtener un conjunto de resultados de una llamada de procedimiento almacenado, si ese procedimiento almacenado devuelve un solo conjunto de resultados y tiene solamente parámetros de entrada. Si el procedimiento almacenado devuelve varios conjuntos de resultados, debe utilizar el método `Statement.execute`. Consulte el tema Obtención de varios conjuntos de resultados de un procedimiento almacenado en una aplicación JDBC para obtener más información.

Para obtener filas de una tabla utilizando una sentencia SELECT con marcadores de parámetros, siga estos pasos:

1. Invoque el método `Connection.prepareStatement` para crear un objeto `PreparedStatement`.
2. Invoque métodos `PreparedStatement.setXXX` para pasar valores a los parámetros de entrada.
3. Invoque el método `PreparedStatement.executeQuery` para obtener la tabla de resultados de la sentencia SELECT en un objeto `ResultSet`.
4. En un bucle, posicione el cursor utilizando el método `ResultSet.next` y recupere datos de cada columna de la fila actual del objeto `ResultSet` utilizando métodos `getXXX`.
5. Invoque el método `ResultSet.close` para cerrar el objeto `ResultSet`.
6. Invoque el método `PreparedStatement.close` para cerrar el objeto `PreparedStatement` cuando termine de utilizar ese objeto.

Por ejemplo, el código siguiente muestra la obtención de filas de la tabla de empleados (EMPLOYEE) para un empleado determinado. Los números que

aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
String empnum, phonenum;
Connection con;
PreparedStatement pstmt;
    ResultSet rs;
...
pstmt = con.prepareStatement(
  "SELECT EMPNO, PHONENO FROM EMPLOYEE WHERE EMPNO=?");
                              // Crear un objeto PreparedStatement     1
pstmt.setString(1,"000010");   // Asignar valor a parámetro de entrada  2

rs = pstmt.executeQuery();    // Obtener tabla de resultados de la consulta  3
while (rs.next()) {           // Posicionar el cursor                        4
 empnum = rs.getString(1);    // Obtener el valor de la primera columna
 phonenum = rs.getString(2);  // Obtener el valor de la primera columna
 System.out.println("Número de empleado = " + empnum +
   "Número de teléfono = " + phonenum);
                              // Imprimir valores de columnas
}
rs.close();                   // Cerrar ResultSet                  5
pstmt.close();                // Cerrar PreparedStatement          6
```

Figura 13. Uso de PreparedStatement.executeQuery

Puede también utilizar el método `PreparedStatement.executeQuery` para sentencias que no tienen marcadores de parámetros. Cuando ejecuta una consulta muchas veces, puede obtener un mejor rendimiento creando la sentencia de SQL en forma de objeto `PreparedStatement`.

Tareas relacionadas:

- "Recuperación de varios conjuntos de resultados de un procedimiento almacenado en una aplicación JDBC" en la página 323

Información relacionada:

- "Comparación del soporte de controlador para las API de JDBC" en la página 407

Utilización de métodos CallableStatement para llamar a procedimientos almacenados

Para llamar a procedimientos almacenados, invoque métodos contenidos en la clase `CallableStatement`. Los pasos básicos son:

1. Invoque el método `Connection.prepareCall` para crear un objeto `CallableStatement`.
2. Invoque los métodos `CallableStatement.setXXX` para pasar valores a los parámetros de entrada (IN).
3. Invoque el método `CallableStatement.registerOutParameter` para indicar qué parámetros son de solo salida (OUT) o cuáles son de entrada y salida (INOUT).
4. Invoque uno de los métodos siguientes para llamar al procedimiento almacenado:

   `CallableStatement.executeUpdate`
   Invoque este método si el procedimiento almacenado no devuelve conjuntos de resultados.

   `CallableStatement.executeQuery`
   Invoque este método si el procedimiento almacenado devuelve un solo conjunto de resultados.

CallableStatement.execute
Invoque este método si el procedimiento almacenado devuelve varios conjuntos de resultados.

5. Si el procedimiento almacenado devuelve conjuntos de resultados, recupere los resultados. Consulte el tema Recuperar varios conjuntos de resultados de un procedimiento almacenado en una aplicación JDBC.
6. Invoque los métodos `CallableStatement.getXXX` para recuperar valores a partir de los parámetros de salida (OUT) o entrada y salida (INOUT).
7. Invoque el método `CallableStatement.close` para cerrar el objeto `CallableStatement` cuando termine de utilizar ese objeto.

El código siguiente muestra la invocación de un procedimiento almacenado que tiene un parámetro de entrada, cuatro parámetros de salida y no devuelve ningún conjunto de resultados (`ResultSet`). Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
int ifcaret;
int ifcareas;
int xsbytes;
String errbuff;
Connection con;
CallableStatement cstmt;
    ResultSet rs;
...
cstmt = con.prepareCall("CALL DSN8.DSN8ED2(?,?,?,?,?)");                1
                    // Crear un objeto CallableStatement
cstmt.setString (1, "DISPLAY THREAD(*)");                               2
                    // Definir parámetro de entrada (mandato de DB2)
cstmt.registerOutParameter (2, Types.INTEGER);                          3
                    // Registrar parámetros de salida
cstmt.registerOutParameter (3, Types.INTEGER);
cstmt.registerOutParameter (4, Types.INTEGER);
cstmt.registerOutParameter (5, Types.VARCHAR);
cstmt.executeUpdate();      // Llamar al procedimiento almacenado  4
ifcaret = cstmt.getInt(2);  // Obtener valores de parámetros de salida  6
ifcareas = cstmt.getInt(3);
xsbytes = cstmt.getInt(4);
errbuff = cstmt.getString(5);
cstmt.close();                                                          7
```

Figura 14. Utilización de métodos CallableStatement para una llamada de procedimiento almacenado con marcadores de parámetros

Tareas relacionadas:

- "Recuperación de varios conjuntos de resultados de un procedimiento almacenado en una aplicación JDBC" en la página 323

Información relacionada:

- "Diferencias de JDBC entre el Controlador JDBC de DB2 Universal y otros controladores JDBC de DB2" en la página 459
- "Comparación del soporte de controlador para las API de JDBC" en la página 407

Manejo de una excepción de SQL cuando se utiliza el Controlador JDBC de DB2 Universal

Al igual que en cualquier programa Java™, el manejo de errores se realiza mediante bloques de código `try/catch`. Los métodos emiten excepciones cuando se producen errores, y el código contenido en el bloque `catch` maneja esas excepciones.

JDBC proporciona la clase `SQLException` para manejar errores. Todos los métodos de JDBC emiten una instancia de `SQLException` cuando se produce un error durante la ejecución del método. De acuerdo con la especificación JDBC, un objeto `SQLException` contiene la información siguiente:
- Un objeto `String` que contiene una descripción del error o el valor nulo
- Un objeto `String` que contiene el SQLSTATE o el valor nulo
- Un valor `int` que contiene un código de error
- Un puntero que indica la ubicación del objeto `SQLException` siguiente o un valor nulo

El Controlador JDBC universal de DB2 proporciona una interfaz `com.ibm.db2.jcc.DB2Diagnosable` por la que se amplía la clase `SQLException`. La interfaz `DB2Diagnosable` le proporciona más información sobre los errores que se producen al acceder a DB2®. Si el controlador JDBC detecta un error, `DB2Diagnosable` le proporciona la misma información que la clase `SQLException` estándar. Sin embargo, si DB2 detecta el error, `DB2Diagnosable` añade los métodos siguientes, que le proporcionan información adicional sobre el error:

getSqlca
Devuelve un objeto `DB2Sqlca` con la información siguiente:
- Un código de error de SQL
- Los valores SQLERRMC
- El valor SQLERRP
- Los valores SQLERRD
- Los valores SQLWARN
- El SQLSTATE

getThrowable
Devuelve el objeto `java.lang.Throwable` que causó la excepción `SQLException` o un valor nulo si ese objeto no existe.

printTrace
Imprime información de diagnóstico.

Estos son los pasos básicos para el manejo de una excepción `SQLException` en un programa JDBC que se ejecuta con el Controlador JDBC universal de DB2:

1. Proporcione al programa acceso a la interfaz `com.ibm.db2.jcc.DB2Diagnosable` y a la clase `com.ibm.db2.jcc.DB2Sqlca`. Puede calificar al completo todas las referencias a esos elementos o puede importarlos:
   ```
   import com.ibm.db2.jcc.DB2Diagnosable;
   import com.ibm.db2.jcc.DB2Sqlca;
   ```
2. Coloque código que pueda generar una excepción `SQLException` en un bloque `try`.
3. En el bloque `catch`, ejecute los pasos siguientes en un bucle:
   a. Determine si ha recuperado la última excepción `SQLException`. Si no la ha recibido, continúe en el paso siguiente.
   b. Determine si existe cualquier otra información sobre DB2 comprobando la existencia de un objeto `DB2Diagnosable`. Si el objeto existe:

1) Opcional: invoque el método `DB2Diagnosable.printTrace` para escribir toda la información sobre `SQLException` en un objeto `java.io.PrintWriter`.
2) Invoque el método `DB2Diagnosable.getThrowable` para determinar si un objeto `java.lang.Throwable` asociado ha causado la excepción `SQLException`.
3) Invoque el método `DB2Diagnosable.getSqlca` para recuperar el objeto DB2Sqlca.
4) Invoque el método `DB2Sqlca.getSqlCode` para recuperar un valor de código de error de SQL.
5) Invoque el método `DB2Sqlca.getSqlErrmc` para recuperar una serie de caracteres que contiene todos los valores SQLERRMC, o invoque el método `DB2Sqlca.getSqlErrmcTokens` para recuperar los valores SQLERRMC dentro de una matriz.
6) Invoque el método `DB2Sqlca.getSqlErrp` para recuperar el valor SQLERRP.
7) Invoque el método `DB2Sqlca.getSqlErrd` para recuperar los valores SQLERRD dentro de una matriz.
8) Invoque el método `DB2Sqlca.getSqlWarn` para recuperar los valores SQLWARN dentro de una matriz.
9) Invoque el método `DB2Sqlca.getSqlState` para recuperar el valor SQLSTATE.
10) Invoque el método `DB2Sqlca.getMessage` para recuperar texto de mensajes de error del servidor de bases de datos.

c. Invoque el método `SQLException.getNextException` para recuperar la excepción `SQLException` siguiente.

El código siguiente muestra cómo obtener información de la versión para DB2 de una excepción `SQLException` que se proporciona con el Controlador JDBC universal de DB2. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
import java.sql.*;               // Importar paquete de la API de JDBC
import com.ibm.db2.jcc.DB2Diagnosable;  // Importar paquetes de DB2     1
import com.ibm.db2.jcc.DB2Sqlca;    // Soporte de SQLException
java.io.PrintWriter printWriter;    // Para volcar toda la
                                    // información sobre SQLException
...
try {                                                                   2
  // Código que podría generar excepciones de SQL
 ...
} catch(SQLException sqle) {
    while(sqle != null) {       // Comprobar si hay más excepciones     3a
                                // Excepciones de SQL para procesar
      //=====> Proceso opcional de errores exclusivos de DB2
      if (sqle instanceof DB2Diagnosable) {                             3b
           // Comprobar si existe información específica de DB2
         com.ibm.db2.jcc.DB2Diagnosable diagnosable =
           (com.ibm.db2.jcc.DB2Diagnosable)sqle;
          diagnosable.printTrace (printWriter, "");                     3b1
          java.lang.Throwable throwable =
            diagnosable.getThrowable();                                 3b2
          if (throwable != null) {
            // Extraer información sobre java.lang.Throwable,
            // tal como mensaje o rastreo de pila.
            ...
          }
          DB2Sqlca sqlca = diagnosable.getSqlca();                      3b3
                                // Obtener objeto DB2Sqlca
          if (sqlca != null) {    // Comprobar que DB2Sqlca no es nulo
            int sqlCode = sqlca.getSqlCode(); // Obtener código de error
                                              // de SQL  3b4
            String sqlErrmc = sqlca.getSqlErrmc();                      3b5
                               // Obtener el valor SQLERRMC completo
            String[] sqlErrmcTokens = sqlca.getSqlErrmcTokens();
                              // También puede recuperar los valores
                             //  SQLERRMC individuales
            String sqlErrp = sqlca.getSqlErrp();                        3b6
                                 // Obtener el valor SQLERRP
            int[] sqlErrd = sqlca.getSqlErrd();     3b7
                                // Obtener campos SQLERRD
            char[] sqlWarn = sqlca.getSqlWarn();    3b8
                                // Obtener campos SQLWARN
            String sqlState = sqlca.getSqlState();                      3b9
                                // Obtener SQLSTATE
            String errMessage = sqlca.getMessage();                     3b10
                                // Obtener mensaje de error

            System.err.println ("Mensaje de error de servidor: " + errMessage);

            System.err.println ("--------------- SQLCA ---------------");
            System.err.println ("Código de error: " + sqlCode);
            System.err.println ("SQLERRMC: " + sqlErrmc);
            for (int i=0; i< sqlErrmcTokens.length; i++) {
              System.err.println ("  token " + i + ": " + sqlErrmcTokens[i]);
            }
```

Figura 15. Proceso de una excepción SQLException cuando se utiliza el Controlador JDBC universal de DB2 (Parte 1 de 2)

```
            System.err.println ( "SQLERRP: " + sqlErrp );
            System.err.println (
              "SQLERRD(1): " + sqlErrd[0] + "\n" +
              "SQLERRD(2): " + sqlErrd[1] + "\n" +
              "SQLERRD(3): " + sqlErrd[2] + "\n" +
              "SQLERRD(4): " + sqlErrd[3] + "\n" +
              "SQLERRD(5): " + sqlErrd[4] + "\n" +
              "SQLERRD(6): " + sqlErrd[5] );
            System.err.println (
              "SQLWARN1: " + sqlWarn[0] + "\n" +
              "SQLWARN2: " + sqlWarn[1] + "\n" +
              "SQLWARN3: " + sqlWarn[2] + "\n" +
              "SQLWARN4: " + sqlWarn[3] + "\n" +
              "SQLWARN5: " + sqlWarn[4] + "\n" +
              "SQLWARN6: " + sqlWarn[5] + "\n" +
              "SQLWARN7: " + sqlWarn[6] + "\n" +
              "SQLWARN8: " + sqlWarn[7] + "\n" +
              "SQLWARN9: " + sqlWarn[8] + "\n" +
              "SQLWARNA: " + sqlWarn[9] );
            System.err.println ("SQLSTATE: " + sqlState);
                                        // Porción de SQLException
          }
        sqle=sqle.getNextException();    // Recuperar excepcion de SQL siguiente   3c
      }
}
```

Figura 15. Proceso de una excepción SQLException cuando se utiliza el Controlador JDBC universal de DB2 (Parte 2 de 2)

Información relacionada:

- "Códigos de error emitidos por el Controlador JDBC de DB2 Universal" en la página 468

Manejo de una excepción de SQL cuando se utiliza el Controlador JDBC de DB2 de tipo 2

Al igual que en cualquier programa Java™, el manejo de errores se realiza mediante bloques de código `try/catch`. Los métodos emiten excepciones cuando se producen errores, y el código contenido en el bloque `catch` maneja esas excepciones.

JDBC proporciona la clase `SQLException` para manejar errores. Todos los métodos de JDBC emiten una instancia de `SQLException` cuando se produce un error durante la ejecución del método. De acuerdo con la especificación JDBC, un objeto `SQLException` contiene la información siguiente:

- Un objeto `String` que contiene una descripción del error o el valor nulo
- Un objeto `String` que contiene el SQLSTATE o el valor nulo
- Un valor `int` que contiene un código de error
- Un puntero que indica la ubicación del objeto `SQLException` siguiente o un valor nulo

Estos son los pasos básicos para manejar una `excepción de SQL` en un programa de JDBC que se ejecuta bajo el Controlador JDBC de DB2® de tipo 2 para Linux, UNIX® y Windows® (Controlador JDBC de DB2 de tipo 2):

1. Coloque código que pueda generar una excepción `SQLException` en un bloque `try`.
2. En el bloque `catch`, ejecute los pasos siguientes en un bucle:

a. Determine si ha recuperado la última excepción SQLException. Si no la ha recibido, continúe en el paso siguiente.
b. Recupere la información de error procedente de SQLException.
c. Invoque el método SQLException.getNextException para recuperar la excepción SQLException siguiente.

El código siguiente muestra un bloque catch que hace uso de la versión de SQLException para DB2 que se proporciona con el Controlador JDBC de DB2 de tipo 2. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
import java.sql.*;           // Importar paquete de la API de JDBC
...
try {
  // Código que podría generar excepciones de SQL
 ...
} catch(SQLException sqle) {
    while(sqle != null) {       // Comprobar si hay más excepciones  1
      System.out.println("Mensaje:  " + sqle.getMessage());          2
      System.out.println("SQLSTATE: " + sqle.getSQLState());
      System.out.println("Código de error SQL: " + sqle.getErrorCode());
      sqle=sqle.getNextException();   // Recuperar excepción de SQL siguiente  3
    }
}
```

Figura 16. Proceso de una excepción SQLException cuando se utiliza el Controlador JDBC universal de DB2

Tareas relacionadas:

- "Manejo de un aviso de SQL cuando se utiliza el Controlador JDBC de DB2 Universal" en la página 312

Manejo de un aviso de SQL cuando se utiliza el Controlador JDBC de DB2 Universal

A diferencia de los errores de SQL, los avisos de SQL no hacen que los métodos de JDBC emitan excepciones. En lugar de ello, las clases Connection, Statement, PreparedStatement, CallableStatement y ResultSet contienen métodos getWarnings, que es necesario invocar después de ejecutar sentencias de SQL para determinar si se han producido avisos de SQL. La invocación de getWarnings hace que se recupere un objeto SQLWarning.Un objeto SQLWarning genérico contiene la información siguiente:

- Un objeto String que contiene una descripción del aviso o el valor nulo
- Un objeto String que contiene el SQLSTATE o el valor nulo
- Un valor int que contiene un código de error
- Un puntero que indica la ubicación del objeto SQLWarning siguiente o un valor nulo

Cuando se utiliza el Controlador JDBC universal de DB2, al igual que ocurre con un objeto SQLException, un objeto SQLWarning puede contener también información específica de DB2®. La información específica de DB2 correspondiente a un objeto SQLWarning es la misma que la información específica de DB2 correspondiente a un objeto SQLException.

Estos son los pasos básicos para recuperar información sobre avisos de SQL:

1. Inmediatamente después de invocar un método por el que se ejecuta una sentencia de SQL, invoque el método `getWarnings` para obtener un objeto `SQLWarning`.
2. Ejecute en bucle los pasos siguientes:
   a. Determine si el objeto `SQLWarning` es nulo. Si no es nulo, continúe en el paso siguiente.
   b. Invoque el método `SQLWarning.getMessage` para obtener la descripción del aviso.
   c. Invoque el método `SQLWarning.getSQLState` para obtener el valor SQLSTATE.
   d. Invoque el método `SQLWarning.getErrorCode` para obtener el valor del código de error.
   e. Si desea obtener información de aviso específica de DB2, ejecute los mismos pasos realizados para obtener información específica de DB2 para un objeto `SQLException`.
   f. Invoque el método `SQLWarning.getNextWarning` para obtener el objeto `SQLWarning` siguiente.

El código siguiente muestra cómo obtener información genérica sobre `SQLWarning`. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
Connection con;
Statement stmt;
    ResultSet rs;
SQLWarning sqlwarn;
...
stmt = con.createStatement();    // Crear un objeto Statement
rs = stmt.executeQuery("SELECT * FROM EMPLOYEE");
                                 // Obtener tabla de resultados de la consulta
sqlwarn = stmt.getWarnings();    // Obtener los avisos generados            1
while (sqlwarn != null) {        // Mientras haya avisos, obtener e         2a
                                 // imprimir información sobre avisos
  System.out.println ("Descripción del aviso: " + sqlwarn.getMessage());  2b
  System.out.println ("SQLSTATE: " + sqlwarn.getSQLState());              2c
  System.out.println ("Código de error: " + sqlwarn.getErrorCode());      2d
  sqlwarn=sqlwarn.getNextWarning();   // Obtener aviso de SQL siguiente   2f
}
```

Figura 17. Proceso de un aviso de SQL

Tareas relacionadas:

- "Manejo de una excepción de SQL cuando se utiliza el Controlador JDBC de DB2 Universal" en la página 308

Manejo de un aviso de SQL cuando se utiliza el Controlador JDBC de DB2 de tipo 2

A diferencia de los errores de SQL, los avisos de SQL no hacen que los métodos de JDBC emitan excepciones. En lugar de ello, las clases `Connection`, `Statement`, `PreparedStatement`, `CallableStatement` y `ResultSet` contienen métodos `getWarnings`, que es necesario invocar después de ejecutar sentencias de SQL para determinar si se han producido avisos de SQL. La invocación de `getWarnings` hace que se recupere un objeto `SQLWarning`.

El Controlador JDBC de DB2® de tipo 2 para Linux, UNIX® y Windows® (Controlador JDBC de DB2 de tipo 2) genera objetos `SQLWarning` genéricos. Un objeto `SQLWarning` genérico contiene la información siguiente:

- Un objeto `String` que contiene una descripción del aviso o el valor nulo
- Un objeto `String` que contiene el SQLSTATE o el valor nulo
- Un valor `int` que contiene un código de error
- Un puntero que indica la ubicación del objeto `SQLWarning` siguiente o un valor nulo

Estos son los pasos básicos para recuperar información sobre avisos de SQL:

1. Inmediatamente después de invocar un método por el que se ejecuta una sentencia de SQL, invoque el método `getWarnings` para obtener un objeto `SQLWarning`.
2. Ejecute en bucle los pasos siguientes:
   a. Determine si el objeto `SQLWarning` es nulo. Si no es nulo, continúe en el paso siguiente.
   b. Invoque el método `SQLWarning.getMessage` para obtener la descripción del aviso.
   c. Invoque el método `SQLWarning.getSQLState` para obtener el valor SQLSTATE.
   d. Invoque el método `SQLWarning.getErrorCode` para obtener el valor del código de error.
   e. Invoque el método `SQLWarning.getNextWarning` para obtener el objeto `SQLWarning` siguiente.

El código siguiente muestra cómo obtener información genérica sobre `SQLWarning`. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
Connection con;
Statement stmt;
    ResultSet rs;
SQLWarning sqlwarn;
...
stmt = con.createStatement();    // Crear un objeto Statement
rs = stmt.executeQuery("SELECT * FROM EMPLOYEE");
                               // Obtener tabla de resultados de la consulta
sqlwarn = stmt.getWarnings();  // Obtener avisos generados                1
while (sqlwarn != null) {     // Mientras haya avisos, obtener e         2a
                             // imprimir información sobre avisos
  System.out.println ("Descripción del aviso: " + sqlwarn.getMessage()); 2b
  System.out.println ("SQLSTATE: " + sqlwarn.getSQLState());             2c
  System.out.println ("Código de error: " + sqlwarn.getErrorCode());     2d
  sqlwarn=sqlwarn.getNextWarning();   // Obtener aviso de SQL siguiente  2f
}
```

Figura 18. Proceso de un aviso de SQL

Tareas relacionadas:

- "Manejo de una excepción de SQL cuando se utiliza el Controlador JDBC de DB2 de tipo 2" en la página 311

Conceptos avanzados de la programación de aplicaciones JDBC

Los temas siguientes contienen información más compleja sobre la escritura de aplicaciones JDBC.

Datos LOB en aplicaciones JDBC con el Controlador JDBC de DB2 Universal

El Controlador JDBC universal de DB2 el soporte de LOB completo en la especificación de JDBC 2.0. Este controlador también incluye soporte para los LOB en métodos adicionales y para tipos de datos adicionales.

Los datos CLOB se envían siempre al servidor de bases de datos en forma de corriente de datos Unicode. El servidor de bases de datos convierte los datos a la página de códigos de destino.

Soporte de localizador de LOB: el Controlador JDBC universal de DB2 puede utilizar localizadores de LOB para recuperar datos de columnas LOB. Para que JDBC utilice localizadores de LOB para recuperar datos de columnas LOB, debe establecer la propiedad `fullyMaterializeLobData` en `false`. Las propiedades se describen en el tema Propiedades del Controlador JDBC de DB2® Universal.

`fullyMaterializeLobData` no tiene ningún efecto para parámetros de procedimiento almacenado ni datos LOB que se recuperan utilizando cursores desplazables. Cuando se utilizan cursores desplazables para recuperar datos de un servidor DB2 UDB en el entorno OS/390® o z/OS™, JDBC utiliza siempre localizadores de LOB para recuperar datos de columnas LOB.

Al igual que en cualquier otro lenguaje, en una aplicación Java un localizador de LOB está asociado a una sola base de datos.. No puede utilizar un localizador de LOB individual para trasladar datos entre dos bases de datos diferentes. Para trasladar datos entre dos bases de datos, es necesario materializar los datos LOB cuando los recupera de una tabla de la primera base de datos y luego insertar esos datos en la tabla de la segunda base de datos.

Métodos adicionales soportados por el Controlador JDBC universal de DB2: además de los métodos descritos en la especificación de JDBC, el Controlador JDBC universal de DB2 incluye soporte de LOB en los métodos siguientes:

- Puede especificar una columna BLOB como argumento de los métodos `ResultSet` siguientes para recuperar datos de una columna BLOB:
  - `getBinaryStream`
  - `getBytes`
- Puede especificar una columna CLOB como argumento de los métodos `ResultSet` siguientes para recuperar datos de una columna CLOB:
  - `getAsciiStream`
  - `getCharacterStream`
  - `getString`
  - `getUnicodeStream`
- Puede utilizar los métodos `PreparedStatement` siguientes para establecer los valores de parámetros que corresponden a columnas BLOB:
  - `setBytes`
  - `setBinaryStream`
- Puede utilizar los métodos `PreparedStatement` siguientes para establecer los valores de parámetros que corresponden a columnas CLOB:

  - setString
  - setAsciiStream
  - setUnicodeStream
  - setCharacterStream
- Puede recuperar el valor de un parámetro CLOB de JDBC utilizando el método CallableStatement siguiente:
  - getString

Restricción sobre el uso de los LOB con el Controlador JDBC universal de DB2: si utiliza Conectividad de tipo 2 universal, no puede invocar un procedimiento almacenado que tenga parámetros DBCLOB OUT o INOUT.

Información relacionada:

- "Propiedades del Controlador JDBC de DB2 Universal" en la página 400
- "Diferencias de JDBC entre el Controlador JDBC de DB2 Universal y otros controladores JDBC de DB2" en la página 459
- "Comparación del soporte de controlador para las API de JDBC" en la página 407

Tipos de datos Java para recuperar o actualizar datos de columnas LOB en aplicaciones JDBC

Cuando la propiedad deferPrepares tiene el valor "true", y el Controlador JDBC universal de DB2 procesa una llamada PreparedStatement.setXXX, el controlador puede necesitar realizar tareas adicionales de proceso para determinar los tipos de datos. Estas tareas adicionales de proceso pueden afectar al rendimiento del sistema.

Cuando el controlador JDBC no puede determinar inmediatamente el tipo de datos de un parámetro que se utiliza con una columna LOB, es necesario elegir un tipo de datos para el parámetro que sea compatible con el tipo de datos LOB.

Parámetros de entrada para columnas BLOB:

Para los parámetros de entrada de columnas BLOB, o parámetros de entrada/salida que se utilizan como entrada para columnas BLOB, puede utilizar uno de los métodos siguientes:

- Utilice una variable de entrada java.sql.Blob, que se corresponde exactamente con una columna BLOB:

  ```
  cstmt.setBlob(parmIndex, blobData);
  ```

- Utilice una llamada CallableStatement.setObject que especifique que el tipo de datos de destino es BLOB:

  ```
  byte[] byteData = {(byte)0x1a, (byte)0x2b, (byte)0x3c};
  cstmt.setObject(parmInd, byteData, java.sql.Types.BLOB);
  ```

- Utilice un parámetro de entrada de tipo java.io.ByteArrayInputStream con una llamada CallableStatement.setBinaryStream. Un objeto java.io.ByteArrayInputStream es compatible con un tipo de datos BLOB. Para esta llamada es necesario especificar la longitud exacta de los datos de entrada:

  ```
  java.io.ByteArrayInputStream byteStream =
    new java.io.ByteArrayInputStream(byteData);
  int numBytes = byteData.length;
  cstmt.setBinaryStream(parmIndex, byteStream, numBytes);
  ```

Parámetros de salida para columnas BLOB:

Para los parámetros de salida de columnas BLOB, o parámetros de entrada/salida que se utilizan como salida para columnas BLOB, puede utilizar uno de los métodos siguientes:

- Utilice la llamada `CallableStatement.registerOutParameter` para especificar que un parámetro de salida es de tipo BLOB. Luego puede recuperar el valor del parámetro e insertarlo en cualquier variable cuyo tipo de datos sea compatible con un tipo de datos BLOB. Por ejemplo, el código siguiente permite recuperar un valor BLOB e insertarlo en una variable byte[]:

```
cstmt.registerOutParameter(parmIndex, java.sql.Types.BLOB);
cstmt.execute();
byte[] byteData = cstmt.getBytes(parmIndex);
```

Parámetros de entrada para columnas CLOB:

Para los parámetros de entrada de columnas CLOB, o parámetros de entrada/salida que se utilizan como entrada para columnas CLOB, puede utilizar uno de los métodos siguientes:

- Utilice una variable de entrada java.sql.Clob, que se corresponde exactamente con una columna CLOB:

```
cstmt.setClob(parmIndex, clobData);
```

- Utilice una llamada `CallableStatement.setObject` que especifique que el tipo de datos de destino es CLOB:

```
String charData = "CharacterString";
cstmt.setObject(parmInd, charData, java.sql.Types.CLOB);
```

- Utilice uno de los tipos siguientes de parámetros de entrada:
  - Un parámetro de entrada `java.io.StringReader` con una llamada `cstmt.setCharacterStream`:

```
java.io.StringReader reader = new java.io.StringReader(charData);
cstmt.setCharacterStream(parmIndex, reader, charData.length);
```

  - Un parámetro `java.io.ByteArrayInputStream` con una llamada `cstmt.setAsciiStream`, para datos ASCII:

```
byte[] charDataBytes = charData.getBytes("US-ASCII");
java.io.ByteArrayInputStream byteStream =
  new java.io.ByteArrayInputStream (charDataBytes);
cstmt.setAsciiStream(parmIndex, byteStream, charDataBytes.length);
```

  Para estas llamadas es necesario especificar la longitud exacta de los datos de entrada.

- Utilice un parámetro de entrada de tipo String con una llamada `cstmt.setString`:

```
cstmt.setString(charData);
```

  Si la longitud de los datos es mayor que 32 KB, el controlador JDBC asigna el tipo de datos CLOB a los datos de entrada.

- Utilice un parámetro de entrada de tipo String con una llamada `cstmt.setObject`, y especifique el tipo de datos de destino como VARCHAR o LONGVARCHAR:

```
cstmt.setObject(parmIndex, charData, java.sql.Types.VARCHAR);
```

  Si la longitud de los datos es mayor que 32 KB, el controlador JDBC asigna el tipo de datos CLOB a los datos de entrada.

Parámetros de salida para columnas CLOB:

Para los parámetros de salida de columnas CLOB, o parámetros de entrada/salida que se utilizan como salida para columnas CLOB, puede utilizar uno de los métodos siguientes:

- Utilice la llamada `CallableStatement.registerOutParameter` para especificar que un parámetro de salida es de tipo CLOB. Luego puede recuperar el valor del parámetro e insertarlo en cualquier variable cuyo tipo de datos sea compatible con un tipo de datos CLOB. Por ejemplo, el código siguiente permite recuperar un valor CLOB e insertarlo en una variable de tipo String:

  ```
  cstmt.registerOutParameter(parmIndex, java.sql.Types.CLOB);
  cstmt.execute();
  String charData = cstmt.getString(parmIndex);
  ```

- Utilice la llamada `CallableStatement.registerOutParameter` para especificar que un parámetro de salida es de tipo VARCHAR o LONGVARCHAR:

  ```
  cstmt.registerOutParameter(parmIndex, java.sql.Types.VARCHAR);
  cstmt.execute();
  String charData = cstmt.getString(parmIndex);
  ```

  Este método solo se debe utilizar si conoce que la longitud de los datos recuperados es menor o igual que 32 KB. En otro caso, los datos se truncan.

Conceptos relacionados:

- "Datos LOB en aplicaciones JDBC con el Controlador JDBC de DB2 Universal" en la página 315

Información relacionada:

- "Tipos de datos de Java, JDBC y SQL" en la página 395

Uso de valores ROWID en JDBC con el Controlador JDBC universal de DB2

DB2® UDB para z/OS® y DB2 UDB para iSeries™ dan soporte al tipo de datos ROWID para una columna de una tabla DB2. Un ROWID es un valor que identifica una fila de una tabla de una forma exclusiva.

Puede utilizar los siguientes de métodos de `ResultSet` para recuperar datos de una columna ROWID:

- `getBytes`
- `getObject`

Para `getObject`, el Controlador JDBC universal de DB2 devuelve una instancia de la clase `com.ibm.db2.jcc.DB2RowID`, exclusiva de DB2.

Puede utilizar los métodos siguientes de `PreparedStatement` para definir un valor de un parámetro asociado a una columna ROWID:

- `setBytes`
- `setObject`

Para `setObject`, utilice el tipo `com.ibm.db2.jcc.Types.ROWID`, exclusivo de DB2, o una instancia de la clase `com.ibm.db2.jcc.DB2RowID` como tipo de destino para los parámetros.

Ejemplo: uso de PreparedStatement.setObject con un tipo de destino com.ibm.db2.jcc.DB2Types.ROWID: para definir el parámetro 1, utilice este formato del método `SetObject`:

```
ps.setObject(1, bytes[], com.ibm.db2.jcc.DB2Types.ROWID);
```

Ejemplo: uso de PreparedStatement.setObject con un tipo de destino com.ibm.db2.jcc.DB2RowID: suponga que rwid es una instancia de `com.ibm.db2.jcc.DB2RowID`. Para definir el parámetro 1, utilice este formato del método `SetObject`:

```
ps.setObject (1, rwid);
```

Para invocar un procedimiento almacenado que está definido con un parámetro de salida ROWID, registre ese parámetro para que sea del tipo `com.ibm.db2.jcc.DB2Types.ROWID`.

Ejemplo: uso de CallableStatement.registerOutParameter con un tipo de parámetro com.ibm.db2.jcc.DB2Types.ROWID: para registrar el parámetro 1 de una sentencia CALL como perteneciente al tipo de datos `com.ibm.db2.jcc.DB2Types.ROWID`, utilice este formato del método `registerOutParameter`:

```
cs.registerOutParameter(1, com.ibm.db2.jcc.DB2Types.ROWID)
```

Información relacionada:

- "Tipos de datos de Java, JDBC y SQL" en la página 395

Tipos diferenciados en aplicaciones JDBC

Un tipo diferenciado es un tipo de datos definido por el usuario cuya representación interna es un tipo de datos SQL incorporado. Un tipo diferenciado se crea ejecutando la sentencia CREATE DISTINCT TYPE de SQL.

En un programa JDBC, puede crear un tipo diferenciado utilizando el método `executeUpdate` para ejecutar la sentencia CREATE DISTINCT TYPE. Puede también utilizar `executeUpdate` para crear una tabla que incluya una columna de ese tipo. Cuando recupera datos de una columna de ese tipo, o actualiza una columna de ese tipo, utiliza identificadores de Java™ con tipos de datos que corresponden a los tipos incorporados en los que se basan los tipos diferenciados.

El ejemplo siguiente crea un tipo diferenciado que está basado en un tipo INTEGER, crea una tabla con una columna de ese tipo, inserta una fila en la tabla y recupera la fila de la tabla:

```
Connection con;
Statement stmt;
    ResultSet rs;
String empNumVar;
int shoeSizeVar;
...
stmt = con.createStatement();    // Crear un objeto Statement
stmt.executeUpdate(
  "CREATE DISTINCT TYPE SHOESIZE AS INTEGER");
                                 // Crear tipo diferenciado
stmt.executeUpdate(
  "CREATE TABLE EMP_SHOE (EMPNO CHAR(6), EMP_SHOE_SIZE SHOESIZE)");
                                 // Crear tabla con tipo diferenciado
stmt.executeUpdate("INSERT INTO EMP_SHOE " +
  "VALUES ('000010', 6)");       // Insertar una fila
rs=stmt.executeQuery("SELECT EMPNO, EMP_SHOE_SIZE FROM EMP_SHOE);
                                 // Crear ResultSet para consulta
while (rs.next()) {
  empNumVar = rs.getString(1);   // Obtener número de empleado
  shoeSizeVar = rs.getInt(2);   // Obtener talla de zapato (shoe size).
                                 // Utilizar int, pues SHOESIZE está
                                 // basado en el tipo INTEGER.
  System.out.println("Número de empleado = " + empNumVar +
    " Talla de zapato = " + shoeSizeVar);
}
rs.close();                     // Cerrar ResultSet
stmt.close();                   // Cerrar Statement
```

Figura 19. Creación y utilización de un tipo diferenciado

Información relacionada:

- "Sentencia CREATE DISTINCT TYPE" en la publicación *Consulta de SQL, Volumen 2*

Puntos de rescate en aplicaciones JDBC

Un punto de rescate de SQL representa el estado que tienen datos y esquemas en un momento determinado dentro de una unidad de trabajo. Existen sentencias de SQL para establecer un punto de rescate, liberar un punto de rescate y para restaurar datos y esquemas al estado representado por el punto de rescate.

El Controlador JDBC universal de DB2 soporta los métodos siguientes para utilizar puntos de rescate:

Connection.setSavepoint() o Connection.setSavepoint(String *nombre*)
: Establece un punto de rescate. Estos métodos devuelven un objeto `Savepoint`, que posteriormente se utiliza en operaciones `releaseSavepoint` o `rollback`.

  Cuando ejecuta cualquiera de estos dos métodos, DB2® ejecuta la modalidad de la sentencia SAVEPOINT que hace uso de la especificación ON ROLLBACK RETAIN CURSORS.

Connection.releaseSavepoint(Savepoint *punto_rescate*)
: Libera el punto de rescate especificado y todos los subsiguientes que haya establecidos.

Connection.rollback(Savepoint *punto_rescate*)
: Retrotrae trabajos hasta el punto de rescate especificado.

DatabaseMetaData.supportsSavepoints()
: Indica si una fuente de datos soporta puntos de rescate.

El ejemplo siguiente muestra cómo establecer un punto de rescate, retrotraer trabajos hasta un punto de rescate y liberar el punto de rescate.

```
Connection con;
Statement stmt;
    ResultSet rs;
String empNumVar;
int shoeSizeVar;
...
con.setAutoCommit(false);        // Desactivar la confirmación automática
stmt = con.createStatement();    // Crear un objeto Statement
stmt.executeUpdate(
  "CREATE DISTINCT TYPE SHOESIZE AS INTEGER");
                                 // Crear tipo diferenciado
con.commit();                    // Confirmar la creación
stmt.executeUpdate(
  "CREATE TABLE EMP_SHOE (EMPNO CHAR(6), EMP_SHOE_SIZE SHOESIZE)");
                                 // Crear tabla con tipo diferenciado
con.commit();                   // Confirmar la creación
stmt.executeUpdate("INSERT INTO EMP_SHOE " +
  "VALUES ('000010', 6)");               // Insertar una fila
Savepoint savept = con.setSavepoint();   // Crear un punto de rescate
...
stmt.executeUpdate("INSERT INTO EMP_SHOE " +
  "VALUES ('000020', 10)");         // Insertar otra fila
conn.rollback(savept);              // Retrotraer trabajo hasta
                                    // el punto situado después
                                    // de la primera inserción
...
con.releaseSavepoint(savept);      // Liberar el punto de rescate
stmt.close();                     // Cerrar el objeto Statement
```

Figura 20. Establecimiento, retrotracción y liberación de un punto de rescate en una aplicación JDBC

Tareas relacionadas:

- "Confirmación o retrotracción de transacciones JDBC" en la página 300

Información relacionada:

- "Comparación del soporte de controlador para las API de JDBC" en la página 407

Recuperación de valores de columnas de identidad en aplicaciones JDBC

Una columna de identidad es una columna de tabla DB2® que proporciona a DB2 una forma de generar automáticamente un valor numérico para cada fila. Puede definir una columna de clave en una sentencia CREATE TABLE o ALTER TABLE especificando la cláusula AS IDENTITY cuando defina una columna que tenga un tipo numérico exacto con una escala de 0 (SMALLINT, INTEGER, BIGINT,DECIMAL con una escala de cero, o un tipo diferenciado basado en uno de estos tipos).

Si está utilizando el Controlador JDBC universal de DB2, puede recuperar columnas de identidad de una tabla DB2 utilizando métodos de JDBC 3.0. En un programa JDBC, las columnas de identidad se conocen como claves generadas automáticamente. Para habilitar la recuperación de claves generadas automáticamente a partir de una tabla, es necesario que al insertar filas indique que deseará recuperar valores de claves generadas automáticamente. Para ello debe

establecer un distintivo en una llamada de método Connection.prepareStatement, Statement.executeUpdate o Statement.execute. La sentencia que se ejecuta debe ser una sentencia INSERT o una sentencia INSERT dentro de SELECT. De lo contrario, el controlador JDBC no tiene en cuenta el parámetro por el que se establece el distintivo.

Para recuperar claves generadas automáticamente de una tabla DB2, debe seguir estos pasos:

1. Utilice uno de los métodos siguientes para indicar que desea obtener claves generadas automáticamente:
   - Si piensa utilizar el método PreparedStatement.executeUpdate para insertar filas, invoque uno de estos formatos del método Connection.prepareStatement para crear un objeto PreparedStatement:

     Utilice este formato para una tabla de un servidor de bases de datos cualesquiera que dé soporte a las columnas de identidad:

     ```
     Connection.prepareStatement(sentencia_sql, Statement.RETURN_GENERATED_KEYS);
     ```

     Utilice este formato solo para una tabla de un servidor de bases de datos cualesquiera que dé soporte a las columnas de identidad y a sentencias INSERT dentro de SELECT:

     ```
     Connection.prepareStatement(sentencia_sql, String [] nombresColumnas);
     ```
   - Si utiliza el método Statement.executeUpdate para insertar filas, invoque uno de estos formatos del método Statement.executeUpdate:

     Utilice este formato para una tabla de un servidor de bases de datos cualesquiera que dé soporte a las columnas de identidad:

     ```
     Statement.executeUpdate(sentencia_sql, Statement.RETURN_GENERATED_KEYS);
     ```

     Utilice este formato solo para una tabla de un servidor de bases de datos cualesquiera que dé soporte a las columnas de identidad y a sentencias INSERT dentro de SELECT:

     ```
     Statement.executeUpdate(sentencia_sql, String [] nombresColumnas);
     ```
   - Si utiliza el método Statement.execute para insertar filas, invoque uno de estos formatos del método Statement.execute:

     Utilice este formato para una tabla de un servidor de bases de datos cualesquiera que dé soporte a las columnas de identidad:

     ```
     Statement.execute(sentencia_sql, Statement.RETURN_GENERATED_KEYS);
     ```

     Utilice este formato solo para una tabla de un servidor de bases de datos cualesquiera que dé soporte a las columnas de identidad y a sentencias INSERT dentro de SELECT:

     ```
     Statement.execute(sentencia_sql, String [] nombresColumnas);
     ```
2. Invoque el método PreparedStatement.getGeneratedKeys o el método Statement.getGeneratedKeys para recuperar un objeto ResultSet que contiene los valores de claves generadas automáticamente.

   El tipo de datos de las claves generadas automáticamente del objeto ResultSet es DECIMAL, con independencia del tipo de datos de la columna correspondiente.

El código siguiente crea una tabla con una columna de identidad, inserta filas en la tabla y recupera valores de claves generadas automáticamente para la columna de identidad. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
Connection con;
Statement stmt;
    ResultSet rs;
java.math.BigDecimal iDColVar;
...
stmt = con.createStatement();    // Crear un objeto Statement

stmt.executeUpdate(
  "CREATE TABLE EMP_PHONE (EMPNO CHAR(6), PHONENO CHAR(4), " +
    "IDENTCOL INTEGER GENERATED ALWAYS AS IDENTITY)");
                              // Crear tabla con columna de identidad
stmt.executeUpdate("INSERT INTO EMP_PHONE " +                          1
  "VALUES ('000010', "5555")",        // Insertar una fila
  Statement.RETURN_GENERATED_KEYS);   // Indicar que desea claves
                                      // generadas automáticamente
rs = stmt.getGeneratedKeys();         // Recuperar el valor de clave  2
                                      // generada automáticamente en
                                      // un conjunto de resultados.
                                      // Se devuelve una sola fila.
                                      // Crear ResultSet para consulta
while (rs.next()) {
  idColVar = rs.getBigDecimal(1);     // Obtener valor de clave generada
                                      // automáticamente
  System.out.println("valor de clave generada automáticamente = " + idColVar);
}
rs.close();                           // Cerrar ResultSet
stmt.close();                         // Cerrar Statement
```

Figura 21. Recuperación de claves generadas automáticamente

Conceptos relacionados:

- "Columnas de identidad" en la página 716

Tareas relacionadas:

- "Uso del método PreparedStatement.executeUpdate para actualizar datos de tablas DB2" en la página 303
- "Utilización del método Statement.executeUpdate para crear y modificar objetos DB2" en la página 301

Información relacionada:

- "Comparación del soporte de controlador para las API de JDBC" en la página 407

Recuperación de varios conjuntos de resultados de un procedimiento almacenado en una aplicación JDBC

Si invoca un procedimiento almacenado que devuelve conjuntos de resultados, es necesario que incluya código para recuperar los conjuntos de resultados. Los pasos que debe emprender dependen de si conoce el número de conjuntos de resultados que se devuelven y el contenido de esos resultados.

Recuperación de un número conocido de conjuntos de resultados:

Siga estos pasos para recuperar conjuntos de resultados cuando conoce su número y contenido:

1. Invoque el método `Statement.execute` o `PreparedStatement.execute` para invocar el procedimiento almacenado. Utilice `PreparedStatement.execute` si el procedimiento almacenado tiene parámetros de entrada.

2. Invoque el método `getResultSet` para obtener el primer conjunto de resultados, que está contenido en un objeto `ResultSet`.
3. En un bucle, posicione el cursor utilizando el método `next` y recupere datos de cada columna de la fila actual del objeto `ResultSet` utilizando métodos `getXXX`.
4. Si existen *n* conjuntos de resultados, repita los pasos siguientes *n*-1 veces:
   a. Invoque el método `getMoreResults` para cerrar el conjunto de resultados actual y apuntar al conjunto de resultados siguiente.
   b. Invoque el método `getResultSet` para obtener el conjunto de resultados siguiente, que está contenido en un objeto `ResultSet`.
   c. En un bucle, posicione el cursor utilizando el método `next` y recupere datos de cada columna de la fila actual del objeto `ResultSet` utilizando métodos `getXXX`.

El código siguiente muestra la recuperación de dos conjuntos de resultados. El primer conjunto de resultados contiene una columna INTEGER y el segundo conjunto de resultados contiene una columna CHAR. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
CallableStatement cstmt;
    ResultSet rs;
    int i;
String s;
...
cstmt.execute();              // Llamar al procedimiento almacenado        1
rs = cstmt.getResultSet();    // Obtener el primer conjunto de resultados  2
while (rs.next()) {           // Posicionar el cursor                      3
 i = rs.getInt(1);            // Recuperar el valor del conjunto de
                              // resultados actual
 System.out.println ("Valor del primer conjunto de resultados = " + i);
                              // Imprimir el valor
}
cstmt.getMoreResults();      // Apuntar al segundo conjunto de resultados   4a
                             // y cerrar el primer conjunto de resultados
rs = cstmt.getResultSet();   // Obtener el segundo conjunto de resultados   4b
while (rs.next()) {          // Posicionar el cursor                        4c
s = rs.getString(1);        // Recuperar el valor del conjunto de
                            //resultados actual
 System.out.println ("Valor del segundo conjunto de resultados = " + s);
                           // Imprimir el valor
}
rs.close();               // Cerrar el conjunto de resultados
cstmt.close();            // Cerrar la sentencia
```

Figura 22. Recuperación de conjuntos de resultados conocidos de un procedimiento almacenado

Recuperación de un número desconocido de conjuntos de resultados:

Para recuperar conjuntos de resultados cuando no conoce su número ni su contenido, debe recuperar conjuntos de resultados hasta que no se recupere ninguno más.. Para cada conjunto de resultados, utilice métodos de `ResultSetMetaData` para determinar su contenido. Consulte el tema Uso de ResultSetMetaData para obtener información sobre un conjunto de resultados para conocer cómo determinar el contenido de un conjunto de resultados.

Después de invocar un procedimiento almacenado, siga estos pasos básicos para recuperar el contenido de conjuntos de resultados cuando desconoce su número.

1. Examine el valor devuelto por la sentencia `execute` mediante la que se invocó el procedimiento almacenado. Si el valor devuelto es `true`, existe al menos un conjunto de resultados, por lo que necesita ir al paso siguiente.
2. Ejecute en bucle los pasos siguientes:
   a. Invoque el método `getResultSet` para obtener un conjunto de resultados, que está contenido en un objeto `ResultSet`. La invocación de este método cierra el conjunto de resultados anterior.
   b. Procese el conjunto de resultados, tal como se indica en el tema Uso de ResultSetMetaData para obtener información sobre un conjunto de resultados.
   c. Invoque el método `getMoreResults` para determinar si existe otro conjunto de resultados. Si `getMoreResults` devuelve `true`, vaya al paso 2a para obtener el conjunto de resultados siguiente.

El código siguiente muestra la recuperación de conjuntos de resultados cuando no se conoce su número ni su contenido. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
CallableStatement cstmt;
    ResultSet rs;
...
boolean resultsAvailable = cstmt.execute();
    // Llamar al procedimiento almacenado
while (resultsAvailable) {
    // Comprobar si hay conjuntos  de resultados     1
 ResultSet rs = cstmt.getResultSet();
    // Obtener un conjunto de resultados           2a
 ...                                // Procesar conjunto de resultados
 resultsAvailable = cstmt.getMoreResults();
    // Buscar conjunto de resultados siguiente  2c
    // (También cierra el conjunto
    // de resultados anterior)
}
```

Figura 23. Recuperación de conjuntos de resultados desconocidos de un procedimiento almacenado

Mantener abiertos los conjuntos de resultados:

En la Figura 23, la invocación de `getMoreResults()` cierra el objeto `ResultSet` que es devuelto por la invocación anterior de `getResultSet`. Sin embargo, si está utilizando el Controlador JDBC universal de DB2, puede invocar el formato 3 de `getMoreResults` para JDBC, que tiene un parámetro que determina si se cierra el conjunto de resultados actual o los conjuntos de resultados abiertos previamente. Este formato de `getMoreResults` requiere utilizar JDK 1.4 o versión posterior.

Puede especificar una de estas constantes:

Statement.KEEP_CURRENT_RESULT
Busca el conjunto de resultados siguiente, pero no cierra el conjunto de resultados actual.

Statement.CLOSE_CURRENT_RESULT
Busca el conjunto de resultados siguiente y cierra el conjunto de resultados actual.

Statement.CLOSE_ALL_RESULTS
Cierra todos los conjuntos de resultados que anteriormente se mantuvieron abiertos.

Por ejemplo, el código mostrado en la Figura 24 mantiene abiertos todos los conjuntos de resultados hasta que se haya recuperado el último conjunto de resultados, y luego los cierra todos.

```
CallableStatement cstmt;
    ResultSet rs;
...
boolean resultsAvailable = cstmt.execute(); // Llamar al procedimiento almacenado
while (resultsAvailable) {               // Comprobar si hay conjuntos de resultados
 ResultSet rs = cstmt.getResultSet();   // Obtener un conjunto de resultados
 ...                                  // Procesar conjunto de resultados
 resultsAvailable = cstmt.getMoreResults(Statement.KEEP_CURRENT_RESULT);
                                // Recuperar el conjunto de resultados
                                // siguiente pero no cerrar el anterior
}
resultsAvailable = cstmt.getMoreResults(Statement.CLOSE_ALL_RESULTS);
                                // Cerrar los conjuntos de resultados
```

Figura 24. Mantener abiertos los conjuntos de resultados recuperados del procedimiento almacenados

Tareas relacionadas:

- "Uso de ResultSetMetaData para obtener información sobre un conjunto de resultados" en la página 326

Uso de ResultSetMetaData para obtener información sobre un conjunto de resultados

En las explicaciones anteriores sobre la recuperación de datos de una tabla o conjunto de resultados de un procedimiento almacenado se suponía que se conoce el número de columnas y tipos de datos de las columnas contenidas en la tabla o conjunto de resultados. Esto no siempre es así, especialmente cuando se recuperan datos de una fuente de datos remota. Cuando escriba programas que obtienen `conjuntos de resultados` desconocidos, es necesario utilizar métodos de `ResultSetMetaData` para determinar las características de los `conjuntos de resultados` antes de poder recuperar datos de ellos.

Los métodos de `ResultSetMetaData` proporcionan los tipos de información siguientes:

- El número de columnas de un `conjunto de resultados`
- El calificador de la tabla subyacente del `conjunto de resultados`
- Información sobre una columna, tal como el tipo de datos, la longitud, la precisión, la escala y la posibilidad de contener nulos
- La indicación de si una columna es de solo lectura

Después de invocar el método `executeQuery` para generar el `conjunto de resultados` de una consulta sobre una tabla, siga estos pasos básicos para determinar el contenido del `conjunto de resultados`:

1. Invoque el método `getMetaData` para el objeto `ResultSet` para crear un objeto `ResultSetMetaData`.
2. Invoque el método `getColumnCount` para determinar el número de columnas del `conjunto de resultados`.
3. Para cada columna del `conjunto de resultados`, ejecute métodos de `ResultSetMetaData` para determinar las características de las columnas.

   Los resultados de `ResultSetMetaData.getColumnName` para una misma definición de tabla pueden diferir, dependiendo de la fuente de datos. Sin embargo, la

información devuelta refleja correctamente la información sobre nombres de columnas que está almacenada en el catálogo de DB2® para esa fuente de datos.

Por ejemplo, el código siguiente muestra cómo determinar los tipos de datos de todas las columnas de la tabla de empleados (EMPLOYEE). Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
String s;
Connection con;
Statement stmt;
    ResultSet rs;
ResultSetMetaData rsmtadta;
int colCount
int mtadtaint;
    int i;
String colName;
String colType;
...
stmt = con.createStatement();    // Crear un objeto Statement
rs = stmt.executeQuery("SELECT * FROM EMPLOYEE");
                            // Obtener conjunto de resultados de la consulta
rsmtadta = rs.getMetaData();  // Crear un objeto ResultSetMetaData   1
colCount = rsmtadta.getColumnCount();                               2
                            // Encontrar número de columnas de EMP
for (i=1; i<= colCount; i++) {                                      3
 colName = rsmtadta.getColumnName();    // Obtener nombre de columna
 colType = rsmtadta.getColumnTypeName();
                                        // Obtener tipo de datos de la columna
 System.out.println("Columna = " + colName +
  " es del tipo de datos " + colType);
                                        // Imprimir el valor de columna
}
```

Figura 25. Uso de métodos de ResultSetMetaData para obtener información sobre un conjunto de resultados

Tareas relacionadas:

- "Utilización de métodos CallableStatement para llamar a procedimientos almacenados" en la página 306
- "Utilización del método Statement.executeQuery para recuperar datos de tablas DB2" en la página 302

Uso de DatabaseMetaData para obtener información sobre una fuente de datos

La interfaz `DatabaseMetaData` contiene métodos para recuperar información sobre una fuente de datos. Estos métodos son útiles cuando escribe aplicaciones genéricas que pueden acceder a diversas fuentes de datos. En estos tipos de aplicaciones, es necesario comprobar si una fuente de datos puede manejar diversas operaciones de base de datos antes de ejecutarlas. Por ejemplo, debe determinar si el controlador de una fuente de datos se encuentra en el nivel JDBC 2.0 antes de invocar métodos de JDBC 2.0 para ese controlador.

Los métodos de `DatabaseMetaData` proporcionan los tipos de información siguientes:

- Características soportadas por la fuente de datos, tales como el nivel SQL de ANSI

- Información específica sobre la fuente de datos, tal como el nivel del controlador
- Límites, tales como el número máximo de columnas que puede tener un índice
- Indicación de si la fuente de datos soporta sentencias de definición de datos (CREATE, ALTER, DROP, GRANT, REVOKE)
- Lista de objetos contenidos en la fuente de datos, tales como tablas, índices o procedimientos
- Indicación de si la fuente de datos soporta diversas funciones de JDBC 2.0, tales como las actualizaciones por lotes o los conjuntos de resultados desplazables (scrollable `ResultSet`)

Si su aplicación se conecta a un servidor DB2® UDB para z/OS™ o OS/390®, es necesario instalar en ese servidor varios procedimientos almacenados para poder invocar algunos métodos de `DatabaseMetaData` que necesitan información del catálogo de DB2. Los procedimientos almacenados son:
- SQLCOLPRIVILEGES
- SQLCOLUMNS
- SQLFOREIGNKEYS
- SQLGETTYPEINFO
- SQLPRIMARYKEYS
- SQLPROCEDURECOLS
- SQLPROCEDURES
- SQLSPECIALCOLUMNS
- SQLSTATISTICS
- SQLTABLEPRIVILEGES
- SQLTABLES
- SQLUDTS

Para DB2 UDB para OS/390 y z/OS, Versión 7, o DB2 UDB para OS/390, Versión 6, los procedimientos almacenados se proporcionan en los PTF. Los PTF se pueden solicitar mediante los canales normales de servicio utilizando los números de PTF siguientes:

Tabla 33. PTF para DB2 Universal Database para z/OS y OS/390

Versión de DB2 Universal Database para z/OS y OS/390	**Número de PTF**
Versión 6	UQ72081 y UQ72082
Versión 7	UQ72083

Consulte al administrador del sistema DB2 UDB para z/OS para conocer si estos procedimientos almacenados están instalados.

Para invocar métodos de `DatabaseMetaData`, debe seguir estos pasos básicos:

1. Cree un objeto `DatabaseMetaData` invocando el método `getMetaData` para la conexión.
2. Invoque métodos de `DatabaseMetaData` para obtener información sobre la fuente de datos.
3. Si el método devuelve un `conjunto de resultados`:
   a. En un bucle, posicione el cursor utilizando el método `next` y recupere datos de cada columna de la fila actual del objeto `ResultSet` utilizando métodos `getXXX`.
   b. Invoque el método `close` para cerrar el objeto `ResultSet`.

Por ejemplo, el código siguiente muestra cómo utilizar métodos de `DatabaseMetaData` para determinar la versión del controlador y obtener una lista de

los procedimientos almacenados disponibles en la fuente de datos. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
Connection con;
DatabaseMetaData dbmtadta;
    ResultSet rs;
int mtadtaint;
String procSchema;
String procName;
...
dbmtadta = con.getMetaData();  // Crear objeto DatabaseMetaData 1
mtadtaint = dmtadta.getDriverVersion();                          2
                  // Comprobar la versión del controlador
System.out.println("Driver version: " + mtadtaint);
rs = dbmtadta.getProcedures(null, null, "%");
                  // Obtener información para todos los procedimientos almacenados
while (rs.next()) {         // Posicionar el cursor              3a
 procSchema = rs.getString("PROCEDURE_SCHEM");
                            // Obtener el esquema del procedimiento
 procName = rs.getString("PROCEDURE_NAME");
                            // Obtener el nombre del procedimiento
 System.out.println(procSchema + "." + procName);
                           // Imprimir nombre de procedimiento calificado
}
rs.close();                // Cerrar ResultSet                   3b
```

Figura 26. Uso de métodos DatabaseMetaData para obtener información sobre una fuente de datos

Información relacionada:

- "Diferencias de JDBC entre el Controlador JDBC de DB2 Universal y otros controladores JDBC de DB2" en la página 459
- "Comparación del soporte de controlador para las API de JDBC" en la página 407

Uso de ParameterMetaData para obtener información sobre los parámetros de un objeto PreparedStatement

El Controlador JDBC universal de DB2 incluye soporte para la interfaz `ParameterMetaData`. La interfaz `ParameterMetaData` contiene métodos obtienen información sobre los marcadores de parámetros de un objeto `PreparedStatement`.

Los métodos de `ParameterMetaData` proporcionan los tipos de información siguientes:

- Los tipos de datos de los parámetros, incluida la precisión y escala de los parámetros decimales.
- Los nombres de tipos de los parámetros, específicos de la base de datos. Para los parámetros que corresponden a columnas de tabla que están definidas con tipos diferenciados, estos nombres son los nombres de los tipos diferenciados.
- Indicación de si los parámetros pueden contener nulos.
- Indicación de si los parámetros son parámetros de entrada o de salida.
- Indicación de si los valores de un parámetro numérico pueden tener signo.
- El nombre de clase Java™ totalmente calificado que el objeto `PreparedStatement.setObject` utiliza cuando define un valor de parámetro.

Para invocar métodos de `ParameterMetaData`, debe seguir estos pasos básicos:

1. Invoque el método `Connection.prepareStatement` para crear un objeto `PreparedStatement`.
2. Invoque el método `PreparedStatement.getParameterMetaData` para obtener un objeto `ParameterMetaData`.
3. Invoque `ParameterMetaData.getParameterCount` para determinar el número de parámetros de `PreparedStatement`.
4. Invoque métodos de `ParameterMetaData` para parámetros individuales.

Por ejemplo, el código siguiente muestra el uso de métodos de `ParameterMetaData` para determinar el número y tipo de datos de los parámetros contenidos en una sentencia UPDATE de SQL. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
Connection con;
ParameterMetaData pmtadta;
int mtadtacnt;
int sqlType;
...
pstmt = con.prepareStatement(
  "UPDATE EMPLOYEE SET PHONENO=? WHERE EMPNO=?");
                     // Crear un objeto PreparedStatement    1
pmtadta = pstmt.getParameterMetaData();                       2
                     // Crear un objeto ParameterMetaData
mtadtacnt = pmtadta.getParameterCount();                      3
                    // Determinar el número de parámetros
System.out.println("Número de parámetros de sentencia: " + mtadtacnt);
for (int i = 1; i <= mtadtacnt; i++) {
  sqlType = pmtadta.getParameterType(i);                      4
                    // Obtener tipo de datos de SQL para cada parámetro
 System.out.println("Tipo de SQL del parámetro " + i " es " + sqlType);
}
...
pstmt.close();                // Cerrar PreparedStatement
```

Figura 27. Uso de métodos de ParameterMetaData para obtener información sobre un objeto PreparedStatement

Información relacionada:

- "Comparación del soporte de controlador para las API de JDBC" en la página 407

Realización de actualizaciones por lotes en aplicaciones JDBC

Los controladores JDBC que soportan JDBC 2.0 y versiones posteriores dan soporte a las actualizaciones por lotes. Mediante las actualizaciones por lotes, en lugar de actualizar una sola fila cada vez de una tabla DB2®, puede hacer que JDBC ejecute varias actualizaciones al mismo tiempo. Las sentencias que se pueden incluir en el mismo lote de actualizaciones se denominan sentencias de *ejecución por lotes*.

Si una sentencia tiene parámetros de entrada o expresiones de lenguaje principal, puede incluir esa sentencia solo en un lote que tenga otras instancias de la misma sentencia. Este tipo de lote se denomina *lote homogéneo*. Si una sentencia carece de parámetros de entrada, puede incluir esa sentencia en un lote solo si las demás sentencias del lote no tienen parámetros de entrada ni expresiones de lenguaje principal. Este tipo de lote se denomina *lote heterogéneo*. Dos sentencias que se puedan incluir en el mismo lote se dice que son *compatibles por lote*.

Utilice los siguientes métodos de Sentencia para crear, ejecutar y eliminar un lote de actualizaciones SQL:
- addBatch
- executeBatch
- clearBatch

Utilice el siguiente método de sentencia preparada y sentencia invocable para crear un lote de parámetros para que una sentencia individual se pueda ejecutar varias veces en un lote, con un conjunto de parámetros diferente para cada ejecución.
- addBatch

Para realizar actualizaciones por lotes utilizando varias sentencias sin parámetros de entrada, siga estos pasos básicos:
1. Inhabilite AutoCommit para el objeto Connection.
2. Invoque el método createStatement para crear un objeto Statement.
3. Para cada sentencia de SQL que desee ejecutar en el lote, invoque el método addBatch.
4. Invoque el método executeBatch para ejecutar el lote de sentencias.
5. Compruebe si se han producido errores. Si no han ocurrido errores:
   a. Obtenga el número de filas afectadas por cada sentencia de SQL a partir de la matriz devuelta por la invocación de executeBatch. Este número no incluye las filas afectadas por activadores o por la aplicación de la integridad referencial.
   b. Invoque el método commit para confirmar los cambios.

Para realizar actualizaciones por lotes utilizando una sola sentencia con varios conjuntos de parámetros de entrada, siga estos pasos básicos:
1. Inhabilite AutoCommit para el objeto Connection.
2. Invoque el método prepareStatement para crear un objeto PreparedStatement para la sentencia de SQL con parámetros de entrada.
3. Para cada conjunto de valores de parámetros de entrada:
   a. Ejecute métodos setXXX para asignar valores a los parámetros de entrada.
   b. Invoque el método addBatch para añadir el conjunto de parámetros de entrada al lote.
4. Invoque el método executeBatch para ejecutar las sentencias con todos los conjuntos de parámetros.
5. Compruebe si se han producido errores. Si no han occurrido errores:
   a. Obtenga el número de filas afectadas por cada ejecución de la sentencia de SQL a partir de la matriz devuelta por la invocación de executeBatch.
   b. Invoque el método commit para confirmar los cambios.

Ejemplo de actualización por lotes: en el siguiente fragmento de código, dos conjuntos de parámetros forman parte de un lote. Luego, una sentencia UPDATE que toma dos parámetros de entrada se ejecuta dos veces, una vez con cada conjunto de parámetros. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
try {
...
  connection con.setAutoCommit(false);                    1
PreparedStatement prepStmt = con.prepareStatement(
    "UPDATE DEPT SET MGRNO=? WHERE DEPTNO=?");            2
  prepStmt.setString(1,mgrnum1);                          3a
  prepStmt.setString(2,deptnum1);
  prepStmt.addBatch();                                    3b

  prepStmt.setString(1,mgrnum2);
  prepStmt.setString(2,deptnum2);
  prepStmt.addBatch();
  int [] numUpdates=prepStmt.executeBatch();              4
  for (int i=0; i < numUpdates.length; i++) {             5a
    if (numUpdates[i] == -2)
      System.out.println("Execution " + i +
        ": unknown number of rows updated");
  else
      System.out.println("Execution " + i +
        "successful: " numUpdates[i] + " rows updated");
  }
  con.commit();                                           5b
} catch(BatchUpdateException b) {
  // process BatchUpdateException
}
```

Figura 28. Realización de una actualización por lotes

Tareas relacionadas:

- "Confirmación o retrotracción de transacciones JDBC" en la página 300

Información relacionada:

- "Diferencias de JDBC entre el Controlador JDBC de DB2 Universal y otros controladores JDBC de DB2" en la página 459

Recuperación de información de una excepción BatchUpdateException

Cuando se produce un error durante la ejecución de una sentencia de un lote de sentencias, el proceso continúa. Pero `executeBatch` emite una excepción `BatchUpdateException`. El objeto `BatchUpdateException` contiene los elementos siguientes:

- Un objeto `String` que contiene una descripción del error o el valor `null`.
- Un objeto `String` que contiene el estado de SQL (SQLSTATE) de la sentencia de SQL anómala, o el valor `null`
- Un valor entero que contiene el código de error o cero
- Una matriz entera de cuentas de actualización para las sentencias de SQL del lote o el valor `null`
- Un puntero que apunta a un objeto `SQLException` o el valor `null`

Se emite una sola excepción `BatchUpdateException` para el lote completo. Como mínimo un objeto `SQLException` está encadenado al objeto `BatchUpdateException`. Los objetos `SQLException` están encadenados en el mismo orden que las sentencias correspondientes que se añadieron al lote. Para ayudarle a asociar los objetos `SQLException` con las sentencias del lote, el campo descriptivo del error de cada objeto `SQLException` comienza con este texto:

`Error para elemento del lote #n:`

n es el número de la sentencia dentro del lote.

Para recuperar información de la excepción `BatchUpdateException`, siga estos pasos:

1. Utilice el método `BatchUpdateException.getUpdateCounts` para determinar el número de filas que actualizó cada sentencia de SQL. `getUpdateCounts` devuelve -2 si no se puede determinar el número de filas actualizadas, o -3 si se produjo un error durante una actualización.
2. Utilice los métodos `getMessage`, `getSQLState` y `getErrorCode` de `SQLException` para obtener la descripción del error, el estado de SQL y el código de error correspondientes al primer error.
3. Utilice el método `BatchUpdateException.getNextException` para obtener una excepción `SQLException` encadenada.
4. Ejecute en bucle las llamadas de método `getMessage`, `getSQLState`, `getErrorCode` y `getNextException` para obtener información sobre una excepción `SQLException` y obtener la siguiente excepción `SQLException`.

Ejemplo de obtención de información de una excepción BatchUpdateException: el fragmento de código siguiente muestra cómo obtener los campos de una excepción `BatchUpdateException` y los objetos `SQLException` encadenados. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
try {
  // Actualizaciones por lotes
} catch(BatchUpdateException buex) {
    System.err.println("Contents of BatchUpdateException:");
    System.err.println(" Update counts: ");
    int [] updateCounts = buex.getUpdateCounts();             1
    for (int i = 0; i < updateCounts.length; i++) {
      System.err.println("  Statement " + i + ":" + updateCounts[i]);
    }
    System.err.println(" Message: " + buex.getMessage());     2
    System.err.println(" SQLSTATE: " + buex.getSQLState());
    System.err.println(" Error code: " + buex.getErrorCode());
    SQLException ex = buex.getNextException();                3
    while (ex != null) {                                      4
      System.err.println("SQL exception:");
      System.err.println(" Message: " + ex.getMessage());
      System.err.println(" SQLSTATE: " + ex.getSQLState());
      System.err.println(" Error code: " + ex.getErrorCode());
      ex = ex.getNextException();
    }
}
```

Figura 29. Recuperación de los campos de una excepción BatchUpdateException

Para obtener información sobre avisos, utilice el método `Statement.getWarnings` para el objeto en el que ejecutó el método `executeBatch`. Luego puede obtener una descripción de error, el estado de SQL y el código de error correspondientes a cada objeto `SQLWarning`.

Restricciones sobre la ejecución de sentencias en un lote:

- Si intenta ejecutar una sentencia SELECT en un lote, se emite una excepción `BatchUpdateException`.
- Un objeto `CallableStatement` que ejecute en un lote puede contener parámetros de salida. Sin embargo, no puede recuperar los valores de los parámetros de salida. Si intenta hacerlo, se emite una excepción `BatchUpdateException`.

- No puede recuperar objetos `ResultSet` de un objeto `CallableStatement` que ejecute en un lote. En ese caso no se emite una excepción `BatchUpdateException`, pero la invocación del método `getResultSet` devuelve un valor nulo.

Tareas relacionadas:

- "Realización de actualizaciones por lotes en aplicaciones JDBC" en la página 330

Características de un conjunto de resultados de JDBC cuando se utiliza el Controlador JDBC de DB2 Universal

Además de avanzar fila a fila por un `conjunto de resultados`, puede ser deseable poder hacer lo siguiente:

- Retroceder o ir directamente a una fila específica
- Actualizar o suprimir filas de un `conjunto de resultados`
- Dejar abierto el `conjunto de resultados` después de una operación COMMIT

Los términos siguientes describen características de un `conjunto de resultados`:

capacidad de desplazamiento
: Capacidad que tiene el cursor de avanzar, retroceder o ir a una fila determinada.

capacidad de actualización
: Capacidad de poder utilizar el cursor para actualizar o suprimir filas. Esta característica no es aplicable a un `conjunto de resultados` devuelto por un procedimiento almacenado, pues un `conjunto de resultados` de un procedimiento almacenado no se puede actualizar.

capacidad de retención
: Capacidad que tiene el cursor de permanecer abierto después de una operación COMMIT.

Un `conjunto de resultados` desplazable de JDBC es equivalente a la tabla de resultados de un cursor DB2® que esté declarado como SCROLL. Un cursor desplazable puede ser *insensible* o *sensible*. Insensible significa que los cambios hechos en la tabla subyacente después de abrir el cursor no son visibles para el cursor. Los cursores insensibles son de solo lectura. Sensible significa lo siguiente:

- Los cambios que el cursor hace en la tabla subyacente son siempre visibles para el cursor.
- Los cambios hechos en la tabla subyacente por otros medios *pueden* ser visibles para el cursor. En DB2, si las filas se recuperan con FETCH INSENSITIVE, los cambios hechos por otros medios no son visibles para el cursor. En DB2, si las filas se recuperan con FETCH SENSITIVE, los cambios hechos por otros medios son visibles para el cursor. En JDBC, la invocación del método `refreshRow` antes de invocar métodos `getXXX` tiene el mismo efecto que FETCH SENSITIVE.

En JDBC, un `conjunto de resultados` puede también ser *estático* o *dinámico*, si el servidor de bases de datos soporta ambos atributos. El usuario determina si los cursores desplazables de un programa son estáticos o dinámicos definiendo la propiedad cursorSensitivity. Consulte el tema Propiedades del Controlador JDBC de DB2 Universal para obtener más información sobre la propiedad cursorSensitivity.

Si un `conjunto de resultados` de JDBC es estático, el tamaño de la tabla de resultados y el orden de las filas en la tabla de resultados no cambian después de

abrir el cursor. Esto significa que no puede hacer inserciones en una tabla de resultados, y si suprime una fila de una tabla de resultados, se produce un hueco por supresión. Puede utilizar el método `rowDeleted` para determinar si la fila actual es un hueco por supresión. Consulte el tema Comparación del soporte de controlador para las API de JDB para obtener una lista completa de los métodos que están soportados para los `conjuntos de resultados`.

Tareas relacionadas:

- "Especificación de las capacidades de actualización, desplazamiento y retención para conjuntos de resultados en aplicaciones de JDBC" en la página 335

Especificación de las capacidades de actualización, desplazamiento y retención para conjuntos de resultados en aplicaciones de JDBC

Para especificar las propiedades de capacidad de desplazamiento, capacidad de actualización y capacidad retención para un `conjunto de resultados`, debe seguir estos pasos:

1. Si la sentencia SELECT por la que se define el `conjunto de resultados` no tiene parámetros de entrada, invoque el método `createStatement` para crear un objeto `Statement`. En otro caso, invoque el método `prepareStatement` para crear un objeto `PreparedStatement`.

   Es necesario especificar modalidades de los métodos `createStatement` o `prepareStatement` que incluyan los parámetros *resultSetType*, *resultSetConcurrency* o *resultSetHoldability* .

   Esta es la modalidad del método `createStatement` que da soporte a la capacidad de desplazamiento y a la capacidad de actualización:

   ```
   createStatement(int resultSetType, int resultSetConcurrency );
   ```

   Esta es la modalidad del método `createStatement` que da soporte a la capacidad de desplazamiento, la capacidad de actualización y la capacidad de retención:

   ```
   createStatement(int resultSetType, int resultSetConcurrency ,
    int resultSetHoldability);
   ```

   Esta es la modalidad del método `prepareStatement` que da soporte a la capacidad de desplazamiento y a la capacidad de actualización:

   ```
   prepareStatement(String sql, int resultSetType,
    int resultSetConcurrency);
   ```

   Esta es la modalidad del método `prepareStatement` que da soporte a la capacidad de desplazamiento, la capacidad de actualización y la capacidad de retención:

   ```
   prepareStatement(String  sql, int resultSetType,
    int resultSetConcurrency, int resultSetHoldability);
   ```

   Consulte la Tabla 34 para ver una lista de los valores válidos para *resultSetType* y *resultSetConcurrency*.

Tabla 34. Combinaciones válidas de valores de resultSetType y resultSetConcurrency para conjuntos de resultados desplazables

Valor de *resultSetType*	**Valor de** *resultSetConcurrency*
`TYPE_FORWARD_ONLY`	`CONCUR_READ_ONLY`
`TYPE_FORWARD_ONLY`	`CONCUR_UPDATABLE`

Tabla 34. Combinaciones válidas de valores de resultSetType y resultSetConcurrency para conjuntos de resultados desplazables (continuación)

Valor de *resultSetType*	**Valor de** *resultSetConcurrency*
TYPE_SCROLL_INSENSITIVE	CONCUR_READ_ONLY
TYPE_SCROLL_SENSITIVE	CONCUR_READ_ONLY
TYPE_SCROLL_SENSITIVE	CONCUR_UPDATABLE

resultSetHoldability tiene dos valores posibles: HOLD_CURSORS_OVER_COMMIT y CLOSE_CURSORS_AT_COMMIT. Cualquiera de estos dos valores se puede especificar con una combinación válida cualquiera de *resultSetConcurrency* y *resultSetHoldability*. El valor que defina prevalece sobre la capacidad de retención por omisión de la conexión.

Restricción: si el conjunto de resultados es desplazable y se utiliza para seleccionar columnas de una tabla en un servidor de DB2 UDB para Linux, UNIX y Windows, la sentencia SELECT que sirve para definir el conjunto de resultados no puede seleccionar columnas que tengan los tipos de datos siguientes:

- LONG VARCHAR
- LONG VARGRAPHIC
- DATALINK
- BLOB
- CLOB
- Un tipo diferenciado que esté basado en cualquiera de los tipos de datos anteriores de esta lista
- Un tipo estructurado

2. Si la sentencia SELECT tiene parámetros de entrada, invoque métodos setXXX para pasar valores a los parámetros de entrada.
3. Invoque el método executeQuery para obtener la tabla de resultados de la sentencia SELECT en un objeto ResultSet.
4. Para cada fila a la que desee acceder:
   a. Posicione el cursor utilizando uno de los métodos listados en la Tabla 35.

Tabla 35. Métodos para posicionar un cursor desplazable en un conjunto de resultados

Método	**Posiciona el cursor**
first()	En la primera fila del conjunto de resultados
last()	En la última fila del conjunto de resultados
next()[1]	En la fila siguiente del conjunto de resultados
previous()[2]	En la fila anterior del conjunto de resultados
absolute(int *n*)[3]	Si *n*>0, en la fila *n* del conjunto de resultados. Si *n*<0 y *m* es el número de filas del conjunto de resultados, en la fila *m*+*n*+*1* del conjunto de resultados.
relative(int *n*)[4,5]	Si *n*>0, en la fila que está situada *n* filas después de la fila actual. Si *n*<0, en la fila que está situada *n* filas antes de la fila actual. Si *n*=0, en la fila actual.
afterLast()	Después de la última fila del conjunto de resultados
beforeFirst()	Antes de la primera fila del conjunto de resultados

Tabla 35. Métodos para posicionar un cursor desplazable en un conjunto de resultados (continuación)

Método	Posiciona el cursor

Notas:

1. Si el cursor está situado antes de la primera fila del conjunto de resultados, este método posiciona el cursor en la primera fila.
2. Si el cursor está situado después de la última fila del conjunto de resultados, este método posiciona el cursor en la última fila.
3. Si el valor absoluto de *n* es mayor que el número de filas del conjunto de resultados, este método posiciona el cursor después de la última fila si *n* es positivo, o antes de la primera fila si *n* es negativo.
4. El cursor debe estar en una fila válida del conjunto de resultados para poder utilizar este método. Si el cursor está antes de la primera fila o después de la última fila, el método emite una excepción de SQL.
5. Suponga que *m* es el número de filas del conjunto de resultados y *x* es la fila actual del conjunto de resultados. Si $n>0$ y $x+n>m$, el controlador posiciona el cursor después de la última fila. Si $n<0$ y $x+n><1$, el controlador posiciona el cursor antes de la primera fila.

b. Si necesita conocer la posición actual del cursor, utilice el método getRow, isFirst, isLast, isBeforeFirst o isAfterLast para obtener esa información.

c. Si para *resultSetType* ha especificado el valor TYPE_SCROLL_SENSITIVE en el paso 1 en la página 335, y necesita ver los valores más recientes de la fila actual, invoque el método refreshRow.

   Recomendación: debido a que la renovación de las filas de un conjunto de resultados puede afectar negativamente al rendimiento de las aplicaciones, debe invocar refreshRow *solo* cuando necesite ver los datos más recientes.

d. Ejecute una o más de las operaciones siguientes:

   - Para recuperar datos de cada columna de la fila actual del objeto ResultSet, utilice métodos getXXX.
   - Para actualizar la fila actual de la tabla subyacente, utilice métodos updateXXX para asignar valores de columna a la fila actual del conjunto de resultados. Luego utilice updateRow para actualizar la fila correspondiente de la tabla subyacente. Si decide no actualizar la tabla subyacente, invoque el método cancelRowUpdates en lugar del método updateRow.

     El valor *resultSetConcurrency* del conjunto de resultados debe ser CONCUR_UPDATABLE para poder utilizar estos métodos.
   - Para suprimir la fila actual de la tabla subyacente, utilice el método deleteRow. La invocación de deleteRow hace que el controlador sustituya la fila actual del conjunto de resultados espacio vacío.

     El valor *resultSetConcurrency* del conjunto de resultados debe ser CONCUR_UPDATABLE para poder utilizar este método.

5. Invoque el método close para cerrar el objeto ResultSet.
6. Invoque el método close para cerrar el objeto Statement o PreparedStatement.

Por ejemplo, el código siguiente recupera todas las filas de la tabla de empleados en orden inverso, y actualiza el número de teléfono para el número de empleado "000010". Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
String s;
Connection con;
Statement stmt;
    ResultSet rs;
...
stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                           ResultSet.CONCUR_UPDATABLE);                    1
                         // Crear un objeto Statement
                         // para un conjunto de resultados
                         // desplazable y actualizable
rs = stmt.executeQuery("SELECT EMPNO FROM EMPLOYEE FOR UPDATE OF PHONENO");
                                       // Crear el conjunto de resultados  3
rs.afterLast();                        // Posicionar el cursor al final
                                       // del conjunto de resultados       4a
while (rs.previous()) {                // Retroceder el cursor
 s = rs.getString("EMPNO");            // Obtener el número de empleado    4d
                                       // (columna 1 de la tabla de
                                       // resultados)
 System.out.println("Employee number = " + s);
                                       // Imprimir el valor de columna
 if (s.compareTo("000010") == 0) {     // Buscar empleado 000010
  updateString("PHONENO","4657");      // Actualizar número de
                                       // teléfono del empleado
  updateRow();                         // Actualizar la fila
 }
}
rs.close();                            // Cerrar ResultSet    5
stmt.close();                          // Cerrar Statement    6
```

Figura 30. Uso de un cursor desplazable

Creación y despliegue de objetos DataSource

A partir de la versión 2.0, JDBC proporciona la interfaz `DataSource` para conectar con una fuente de datos. La utilización de la interfaz `DataSource` es la forma preferida de conectar con una fuente de datos. La utilización de la interfaz `DataSource` comprende dos etapas:

- Crear y desplegar objetos `DataSource`. Esto es realizado habitualmente por el administrador del sistema, utilizando una herramienta tal como WebSphere® Application Server.
- Utilizar objetos `DataSource` para crear una conexión. Esto se realiza en el programa de aplicación.

Este tema contiene información necesaria para que el propio usuario pueda crear y desplegar objetos `DataSource`.

El Controlador JDBC universal de DB2 proporciona las siguientes implementaciones de `DataSource`:

- `com.ibm.db2.jcc.DB2SimpleDataSource`, que no da soporte a la agrupación de conexiones. Puede utilizar esta implementación con Conectividad de tipo 2 universal o Conectividad de tipo 4 universal.

El Controlador JDBC de DB2® de tipo 2 proporciona las siguientes implementaciones de `DataSource`:

- `COM.ibm.db2.jdbc.DB2DataSource`, que tiene soporte interno para la agrupación de conexiones. Con esta implementación, la agrupación de conexiones es gestionada internamente y es transparente para la aplicación.

- `COM.ibm.db2.jdbc.DB2XADataSource`, que no tiene soporte interno para las transacciones distribuidas y la agrupación de conexiones. Con esta implementación, el propio usuario debe gestionar las transacciones distribuidas y la agrupación de conexiones, ya sea escribiendo su propio código de programación o utilizando una herramienta tal como WebSphere Application Server.

Cuando crea y despliega un objeto `DataSource`, debe efectuar estas tareas:

1. Crear una instancia de la implementación apropiada de `DataSource`.
2. Definir las propiedades del objeto `DataSource`.
3. Registrar el objeto con el servicio de asignación de nombres de JNDI (Java™ Naming and Directory Interface).

El ejemplo de la Figura 31 muestra cómo realizar esas tareas.

```
import java.sql.*;    // Base JDBC
import javax.naming.*;    // Servicios de asignación de nombres de JNDI
import javax.sql.*;       // Las API de extensión estándar de JDBC 2.0
import com.ibm.db2.jcc.*; // Implementación para DB2 de las API de
                          // extensión estándar de JDBC 2.0

DB2SimpleDataSource db2ds = new com.ibm.db2.jcc.DB2SimpleDataSource(); 1

db2ds.setDatabaseName("db2loc1");                                      2
db2ds.setDescription("Our Sample Database");
db2ds.setUser("john");
db2ds.setPassword("db2");
:
Context ctx=new InitialContext();                                      3
Ctx.bind("jdbc/sampledb",db2ds);                                       4
```

Figura 31. Ejemplo de creación y despliegue de un objeto DataSource

1 Crea una instancia de la clase DB2SimpleDataSource.

2 Esta sentencia y las tres sentencias siguientes definen valores para propiedades del objeto `DB2SimpleDataSource`.

3 Crea un contexto para su utilización por JNDI.

4 Asocia el objeto db2ds de `DBSimple2DataSource` con el nombre lógico jdbc/sampledb. Una aplicación que haga uso de este objeto puede hacer referencia a él utilizando el nombre jdbc/sampledb.

Información relacionada:

- "Propiedades del Controlador JDBC de DB2 Universal" en la página 400

Soporte de redireccionamiento del cliente del Controlador JDBC universal de DB2

La gestión de anomalías es la capacidad de un servidor para asumir operaciones cuando falla otro servidor. El soporte de redireccionamiento del cliente del Controlador JDBC universal de DB2 proporciona soporte para la función de gestión de anomalías en un entorno DB2® UDB para Linux, UNIX® y Windows®. Esto permite que un cliente de DB2 UDB para Linux, UNIX y Windows se recupere de un error de comunicaciones cuando el cliente está conectado a una base de datos de DB2 UDB para Linux, UNIX y Windows. Cuando se produce un error de comunicaciones, el soporte de redireccionamiento del cliente del

Controlador JDBC universal de DB2 hace que la conexión subyacente se redireccione hacia una ubicación alternativa donde reside una réplica de gestión de anomalías de la base de datos. Cuando se mantiene una conexión mediante un redireccionamiento del cliente, se emite una excepción para indicar al usuario que se ha producido un redireccionamiento, y se retrotrae la transacción.

El soporte de redireccionamiento del cliente del Controlador JDBC universal de DB2 solo está disponible para conexiones que hacen uso de la interfaz `javax.sql.DataSource`.

La información de conectividad correspondiente a la ubicación alternativa se proporciona a clientes Java™ mediante la propiedad activeServerListJNDIName de la instancia primaria de la `DataSource` de JDBC. activeServerListJNDIName identifica una referencia de JNDI a una instancia de `DB2ActiveServerList` contenida en un depósito JNDI de información del servidor alternativo.

`DB2ActiveServerList` es un bean Java serializable con dos propiedades: `alternateServerName` y `alternatePortNumber`. Para cada propiedad se definen los métodos `getXXX` y `setXXX`. El bean Java tiene este aspecto:

```
package com.ibm.db2.jcc;
public class DB2ActiveServerList implements java.io.Serializable,
  javax.naming.Referenceable
{
  public String[] alternateServerName;
  public synchronized void
    setAlternateServerName(String[] alternateServer);
  public String[] getAlternateServerName();
  public int[] alternatePortNumber;
  public synchronized void
    setAlternatePortNumber(int[] alternatePortNumberList);
  public int[] getAlternatePortNumber();
}
```

Se propaga dinámicamente información sobre servidores alternativos desde el servidor al cliente cuando éste emite un CONNECT o CONNECT RESET. Esta información sobre servidores alternativos propagada dinámicamente se almacena en memoria de controlador global, y también se actualiza en el almacén de servidores DB2 activos de JNDI. El Controlador JDBC universal de DB2 intenta propagar la información actualizada a la JNDI alternativa después de la gestión de anomalías.

Se configura una conexión de gestión de anomalías recién establecida utilizando las propiedades de la `DataSource` original, salvo en lo que respecta al nombre del servidor y el número de puerto. Además, los registros especiales de DB2 que se modificaron durante la conexión original se restablecen en la conexión de gestión de anomalías.

Cuando se produce un error de comunicaciones, el Controlador JDBC universal de DB2 primero intenta recuperar el servidor original. La reconexión con el servidor original se denomina recuperación. Si la recuperación falla, el controlador intenta conectar con la ubicación alternativa (gestión de anomalías). Después de restablecer una conexión de gestión de anomalías o de recuperación, el controlador emite una excepción `java.sql.SQLException` a la aplicación y devuelve el SQLCODE -4498 para indicar que se ha producido una gestión de anomalías o una recuperación y que la transacción ha fallado. La aplicación puede entonces reintentar su transacción.

*Método de configuración del servidor alternativo:*Utilice JNDI para configurar el servidor alternativo. Esto supone los pasos siguientes:

1. Defina el entorno para un contexto inicial. Para ello puede crear un archivo jndi.properties y añadir su nombre a la variable CLASSPATH.

   Ejemplo: archivo jndi.properties:

   ```
   java.naming.factory.initial=com.sun.jndi.fscontext.RefFSContextFactory
   java.naming.provider.url=file:/tmp
   ```

2. Cree una instancia de `DB2ActiveServerList` y vincule esa instancia con el registro de JNDI.

   Ejemplo: el código siguiente crea una instancia de `DB2ActiveServerList` y vincula esa instancia con el registro de JNDI:

   ```
   // Crear un contexto inicial para operaciones de asignación de nombres
   InitialContext registry = new InitialContext();
   // Crear un objeto DB2ActiveServerList
   DB2ActiveServerList address = new DB2ActiveServerList();
   // Definir el número de puerto y nombre de servidor para el servidor alternativo
   int[] a = {50000};
   String[] s = {"mvs3.sj.ibm.com"};
   address.setActivePortNumber(a);
   address.setActiveServerName(s);
   // Vincular la instancia de DB2ActiveServerList con el registro de JNDI
   registry.rebind("jdbc/alternate", address);
   ```

3. Asigne el nombre lógico del objeto `DB2ActiveServerList`, donde reside la información de ubicación del servidor alternativo, a la propiedad activeServerListJNDIName de la `DataSource` original.

   Ejemplo: el código siguiente asigna el nombre lógico del objeto `DB2ActiveServerList` a la propiedad activeServerListJNDIName de la instancia de `DataSource` denominada datasource:

   ```
   datasource.setActiveServerListJNDIName("jdbc/alternate");
   ```

Información relacionada:

- "Propiedades del Controlador JDBC de DB2 Universal" en la página 400
- "Resumen de las extensiones del Controlador JDBC de DB2 Universal para JDBC" en la página 446

Suministro de información ampliada sobre el cliente al servidor DB2 con el Controlador JDBC universal de DB2

El Controlador JDBC universal de DB2 proporciona métodos exclusivos de DB2® que el usuario puede utilizar para proporcionar información adicional sobre el cliente al servidor. Esta información se puede utilizar con fines contables o para gestionar la carga de trabajo. La información se envía al servidor DB2 cuando la aplicación ejecuta una acción por la que se accede al servidor, tal como ejecutar SQL.

Los métodos aparecen listados en la Tabla 36.

Tabla 36. Métodos que proporcionan información sobre el cliente al servidor DB2

Método	Información proporcionada
setDB2ClientUser	Nombre de usuario para una conexión
setDB2ClientWorkstation	Nombre de estación de trabajo del cliente para una conexión

Tabla 36. Métodos que proporcionan información sobre el cliente al servidor DB2 (continuación)

Método	Información proporcionada
setDB2ClientApplicationInformation	Nombre de la aplicación que está trabajando con una conexión
setDB2ClientAccountingInformation	Información contable

Para establecer el suministro de información ampliada:
1. Cree un objeto `Connection`.
2. Convierta el objeto `java.sql.Connection` en un objeto `com.ibm.db2.jcc.DB2Connection`.
3. Invoque cualquiera de los métodos mostrados en la Tabla 36 en la página 341.
4. Ejecute una sentencia de SQL para hacer que la información se envíe al servidor DB2.

El código siguiente ejecuta los pasos anteriores para pasar un nombre de usuario y un nombre de estación de trabajo al servidor DB2. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
public class ClientInfoTest {
  public static void main(String[] args) {
    String url = "jdbc:db2://sysmvs1.stl.ibm.com:5021/san_jose";
try {
      Class.forName("com.ibm.db2.jcc.DB2Driver");
      String user = "db2adm";
      String password = "db2adm";
      Connection con = DriverManager.getConnection(url,          1
        user, password);
      if (conn instanceof DB2Connection) {
        DB2Connection db2conn = (DB2Connection) conn;            2
        db2conn.setDB2ClientUser("Michael L Thompson");          3
        db2conn.setDB2ClientWorkstation("sjwkstn1");
        // Ejecutar SQL para forzar el envío de información
        // ampliada sobre el cliente al servidor
        conn.prepareStatement("SELECT * FROM SYSIBM.SYSDUMMY1"
          + "WHERE 0 = 1").executeQuery();                       4
    }
    } catch (Throwable e) {
        e.printStackTrace();
      }
  }
}
```

Figura 32. Ejemplo de la transferencia de información ampliada sobre el cliente a un servidor DB2

Información relacionada:

- "Resumen de las extensiones del Controlador JDBC de DB2 Universal para JDBC" en la página 446

Capítulo 16. Programación de aplicaciones SQLJ

Las secciones siguientes contienen información sobre la escritura de aplicaciones SQLJ.

Conceptos básicos de la programación de aplicaciones SQLJ

Los temas siguientes contienen información básica sobre la escritura de aplicaciones SQLJ.

Pasos básicos para escribir una aplicación SQLJ

La escritura de una aplicación JDBC es muy similar a la escritura de una aplicación SQL en cualquier otro lenguaje: en general, es necesario seguir los pasos siguientes:

- Importar los paquetes Java™ donde están contenidos los métodos de SQLJ y JDBC.
- Declarar variables para enviar datos a tablas DB2® o recuperarlos de ellas.
- Conectar con una fuente de datos.
- Ejecutar sentencias de SQL.
- Manejar los errores y avisos de SQL.
- Desconectar de la fuente de datos.

Aunque las tareas necesarias son similares a las que se ejecutan para otros lenguajes, la forma de ejecutarlas y el orden de ejecución es algo diferente.

La Figura 33 en la página 344 es un programa sencillo que muestra la ejecución de cada tarea.

```
import sqlj.runtime.*;                                  1
import java.sql.*;

#sql context EzSqljCtx;                                 3a
#sql iterator EzSqljNameIter (String LASTNAME);         4a

public class EzSqlj {
  public static void main(String args[])
    throws SQLException
  {
    EzSqljCtx ctx = null;
    String URLprefix = "jdbc:db2:";
    String url;
    url = new String(URLprefix + args[0]);
              // El nombre de ubicación es un parámetro de entrada
    String hvmgr="000010";                              2
    String hvdeptno="A00";
    try {                                               3b
      Class.forName("com.ibm.db2.jcc.DB2Driver");
    } catch (Exception e)
    {
      throw new SQLException("Error en EzSqlj: no se pudo cargar controlador");
    }
    try
    {
       System.out.println("Se va a conectar utilizando el url: " + url);
       Connection con0 = DriverManager.getConnection(url);      3c
                                  // Crear una conexión JDBC
       con0.setAutoCommit(false);  // Desactivar la confirmación automática
       ctx = new EzSqljCtx(con0);                       3d

    try
       {
         EzSqljNameIter iter;
         int count=0;

         #sql [ctx] iter =
           {SELECT LASTNAME FROM EMPLOYEE};                     4b
                        // Crear tabla de resultados de SELECT
         while (iter.next()) {                                  4c
           System.out.println(iter.LASTNAME()); // Obtener filas de tabla de resultados
            count++;
         }
         System.out.println("Recuperadas " + count + " filas de datos");
       }
```

Figura 33. Aplicación SQLJ sencilla (Parte 1 de 2)

```
      catch( SQLException e )                                    5
      {
         System.out.println ("**** Excepción de SQL de SELECT...");
         while(e!=null) {
           System.out.println ("Mensaje de error: " + e.getMessage());
           System.out.println ("SQLSTATE: " + e.getSQLState());
           System.out.println ("Código de error: " + e.getErrorCode());
           e = e.getNextException(); // Buscar excepciones encadenadas
         }
      }
   catch (Exception e)
      {
         System.out.println("**** Excepción no de SQL  = " + e);
         e.printStackTrace();
      }
   try
      {
        #sql [ctx]                              4d
          {UPDATE DEPARTMENT SET MGRNO=:hvmgr
             WHERE DEPTNO=:hvdeptno};
        // Actualizar datos para un departamento              6
        #sql [ctx] {COMMIT}; // Confirmar actualización
      }
   catch (SQLException e)
      {
         System.out.println ("**** Excepción de SQL de UPDATE...");
         System.out.println ("Mensaje de error: " + e.getMessage() + ".  SQLSTATE=" +
           e.getSQLState() + "Código de error=" + e.getErrorCode());
         e.printStackTrace();
      }
   catch (Exception e)
      {
         System.out.println("**** Excepción no de SQL  = " + e);
         e.printStackTrace();
      }
      iter.close();        // Cerrar el iterador
      ctx.close();                                          7
 }
   catch (SQLException e)
 {
         System.out.println ("**** Excepción de SQL ...");
         System.out.println ("Mensaje de error: " + e.getMessage() + ".  SQLSTATE=" +
         e.getSQLState() + "Código de error=" + e.getErrorCode());
    e.printStackTrace();
 }
   catch (Exception e)
 {
   System.out.println("**** Excepción no de SQL  = " + e);
   e.printStackTrace();
 }

}
```

Figura 33. Aplicación SQLJ sencilla (Parte 2 de 2)

Nota para la Figura 33 en la página 344:

1 Estas sentencias importan el paquete `java.sql`, que contiene la API básica de JDBC, y el paquete `sqlj.runtime`, que contiene la API de SQLJ. Para obtener información sobre otros paquetes o clases a los que puede necesitar acceder, consulte el tema Acceso a paquetes Java para el soporte de SQLJ.

2 Las variables hvmgr y hvdeptno de tipo `String` son *identificadores de lenguaje principal*, que son equivalentes a variables de lenguaje principal de DB2. Consulte el tema Declarar variables en aplicaciones SQLJ para obtener más información.

3a, **3b**, **3c** y **3d** Estas sentencias muestran cómo conectar con una fuente de datos utilizando una de las tres técnicas disponibles. Consulte el tema Conectar a una fuente de datos utilizando SQLJ para obtener más información.

4a , **4b** , **4c** y **4d**
: Estas sentencias muestran cómo ejecutar sentencias de SQL en SQLJ. La sentencia 4a es el equivalente de SQLJ para declarar un cursor de SQL. Las sentencias 4b y 4c son un equivalente de SQLJ para ejecutar sentencias FETCH de SQL. La sentencia 4d es el equivalente de SQLJ para ejecutar una sentencia UPDATE de SQL. Para obtener más información, consulte el tema Ejecución de SQL en una aplicación SQLJ.

5
: Este bloque `try/catch` muestra el uso de la clase `SQLException` para el manejo de errores de SQL. Para obtener más información sobre el manejo de errores de SQL, consulte el tema Manejo de errores en una aplicación SQLJ. Para obtener más información sobre el manejo de avisos de SQL, consulte el tema Manejo de avisos de SQL en una aplicación SQLJ.

6
: Esto es un ejemplo de un comentario. Para conocer las reglas sobre la inclusión de comentarios en programas SQLJ, consulte el tema Inclusión de comentarios en una aplicación SQLJ.

7
: Esta sentencia cierra la conexión con la fuente de datos. Consulte el tema Cierre de la conexión con la fuente de datos en una aplicación SQLJ.

Conceptos relacionados:

- "Paquetes Java para el soporte SQLJ" en la página 346
- "Variables en aplicaciones SQLJ" en la página 347
- "Sentencias de SQL en una aplicación SQLJ" en la página 356

Tareas relacionadas:

- "Conexión a una fuente de datos utilizando SQLJ" en la página 349

Paquetes Java para el soporte SQLJ

Para ejecutar sentencias de SQLJ o invocar métodos de JDBC en un programa de SQLJ necesita poder acceder a paquetes Java™ donde reside el soporte para esas sentencias. Puede hacerlo importando los paquetes o clases específicas, o bien utilizando nombres de clase totalmente calificados. Puede necesitar los paquetes o clases siguientes para su programa de SQLJ:

sqlj.runtime
: Contiene la API de ejecución de SQLJ.

java.sql
: Contiene la API básica de JDBC.

com.ibm.db2.jcc
: Contiene la implementación de JDBC y SQLJ específica de DB2®.

javax.naming
: Contiene clases e interfaces para JNDI (Java Naming and Directory Interface), que a menudo se utiliza para implementar una fuente de datos (`DataSource`).

javax.sql
: Contiene extensiones estándar de JDBC 2.0.

Conceptos relacionados:

- "Pasos básicos para escribir una aplicación SQLJ" en la página 343

Variables en aplicaciones SQLJ

En los programas DB2® escritos en otros lenguajes, se utilizan variables de lenguaje principal para pasar datos entre el programa de aplicación y DB2. En los programas SQLJ se utilizan *expresiones de lenguaje principal*. Una expresión de lenguaje principal puede ser un simple identificador Java™ o una expresión compleja. Cada expresión de lenguaje principal debe comenzar con dos puntos (:) cuando se utiliza en una sentencia de SQL. Las expresiones de lenguaje principal distinguen entre letras mayúsculas y minúsculas.

Para el Controlador JDBC universal de DB2, un identificador Java puede tener cualquiera de los tipos de datos que se listan en la columna de tipos de datos Java de tipos de datos Java, JDBC y SQLJ. Los tipos de datos que se especifican en un iterador pueden ser cualquiera de los tipos de la columna de tipos de datos Java de tipos de datos Java, JDBC y SQLJ.

Una expresión compleja es un elemento de matriz o expresión Java cuya evaluación da como resultado un valor individual. Las expresiones complejas contenidas en una cláusula de SQLJ deben estar delimitadas por paréntesis.

Los ejemplos siguientes muestran cómo utilizar expresiones de lenguaje principal.

Ejemplo: declaración de un identificador Java y su utilización en una sentencia SELECT:

En este ejemplo, la sentencia que comienza con `#sql` tiene la misma función que una sentencia SELECT en otros lenguajes de programación. Esta sentencia asigna el identificador Java `empname` al apellido del empleado cuyo número de empleado es 000010.

```
String empname;
...
#sql [ctxt]
  {SELECT LASTNAME INTO :empname FROM EMPLOYEE WHERE EMPNO='000010'};
```

Ejemplo: declaración de un identificador Java y su utilización en una llamada de procedimiento almacenado:

En este ejemplo, la sentencia que comienza con `#sql` tiene la misma función que una sentencia CALL de SQL en otros lenguajes de programación. Esta sentencia utiliza el identificador Java `empno` como parámetro de entrada del procedimiento almacenado `A`. El valor `IN`, que precede a `empno`, especifica que `empno` es un parámetro de entrada. El calificador que indica cómo se utiliza el parámetro (IN, OUT o INOUT) debe coincidir con el valor correspondiente contenido en la definición de parámetro que especificó en la sentencia CREATE PROCEDURE para el procedimiento almacenado.

```
String empno = "0000010";
...
#sql [ctxt] {CALL A (:IN empno)};
```

Ejemplo: uso de una expresión compleja como identificador de lenguaje principal:

Este ejemplo utiliza la expresión compleja (((int)yearsEmployed++/5)*500) como expresión de lenguaje principal.

```
#sql [ctxt] {UPDATE EMPLOYEE
  SET BONUS=:(((int)yearsEmployed++/5)*500) WHERE EMPNO=:empID};
```

SQLJ ejecuta las acciones siguientes cuando procesa una expresión de lenguaje principal compleja:

- Evalúa de izquierda a derecha la expresión de lenguaje principal antes de asignar su valor a DB2.
- Evalúa los efectos secundarios, tales como operaciones con operadores sufijos, de acuerdo con las reglas normales de Java. Todas las expresiones de lenguaje principal se evalúan antes de pasar sus valores a DB2.
- Utiliza reglas de Java para el redondeo y truncamiento de datos.

Por tanto, si el valor de yearsEmployed es 6 antes de ejecutar la sentencia UPDATE, el valor que la sentencia UPDATE asigna a la columna BONUS es ((int)6/5)*500, lo que equivale a 500. Después de asignar 500 a BONUS, el valor de yearsEmployed se incrementa.

Restricciones para nombres de variables: existen dos cadenas de caracteres con significados especiales en los programas SQLJ. Tenga en cuenta las restricciones siguientes cuando utilice esas cadenas de caracteres en sus programas SQLJ:

- La cadena __sJT_ es un prefijo reservado que se utiliza para los nombres de variables generados por SQLJ. No comience los siguientes tipos de nombres con __sJT_:
  - Nombres de expresiones de lenguaje principal
  - Nombres de variables Java que están declaradas en bloques en los que se incluyen sentencias ejecutables de SQL
  - Nombres de parámetros para métodos que contienen sentencias ejecutables de SQL
  - Nombres de campos en clases que contienen sentencias ejecutables de SQL, o en clases con subclases o clases incluidas que contienen sentencias ejecutables de SQL
- La cadena _SJ es un sufijo reservado que se utiliza para archivos de recursos y clases que han sido creados por SQLJ. Evite utilizar la cadena _SJ en nombres de clases y nombres de archivos fuente de entrada.

Conceptos relacionados:

- "Pasos básicos para escribir una aplicación SQLJ" en la página 343

Información relacionada:

- "Tipos de datos de Java, JDBC y SQL" en la página 395

Comentarios en una aplicación SQLJ

Para documentar su programa es necesario que incluya comentarios. Para ello, utilice comentarios de Java™. Los comentarios de Java están indicados por /* */ o //. Puede incluir comentarios de Java fuera de las cláusulas de SQLJ, en los lugares permitidos por el lenguaje Java. Dentro de una cláusula de SQLJ, puede utilizar comentarios de Java solo dentro de expresiones de lenguaje principal.

Conceptos relacionados:

- "Pasos básicos para escribir una aplicación JDBC" en la página 287

Conexión a una fuente de datos utilizando SQLJ

En una aplicación SQLJ, al igual que en cualquier otra aplicación de DB2®, debe estar conectado a un servidor de bases de datos para poder ejecutar sentencias de SQL. En SQLJ, como en JDBC, un servidor de bases de datos se denomina *fuente de datos*.

Puede utilizar una de las cinco técnicas siguientes para conectar con una fuente de datos:

- Cree explícitamente una conexión utilizando la interfaz `DriverManager` de JDBC. Puede hacer esto de dos maneras.
- Cree explícitamente una conexión utilizando la interfaz `DataSource` de JDBC. Puede hacer esto de dos maneras.
- Cree implícitamente una conexión.

Técnica de conexión 1: esta técnica utiliza la interfaz `DriverManager` de JDBC como mecanismo subyacente para crear la conexión. Puede utilizar esta técnica con un nivel cualquiera del controlador JDBC.

1. Ejecute una *cláusula de declaración de conexión* de SQLJ.

   Esto genera una *clase de contexto de conexión*. Esta es la forma más simple de la cláusula de declaración de conexión:

   ```
   #sql context nombre_clase_contexto;
   ```

   El nombre de la clase de contexto de conexión generada es *nombre_clase_contexto*.
2. Cargue un controlador JDBC invocando el método `Class.forName`:
   - Para el Controlador JDBC universal de DB2, invoque `Class.forName` de esta manera:

     ```
     Class.forName("com.ibm.db2.jcc.DB2Driver");
     ```
   - Para el Controlador JDBC de DB2 de tipo 2, invoque `Class.forName` de esta manera:

     ```
     Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");
     ```
3. Invoque el constructor para la clase de contexto de conexión que creó en el paso 1.

   Esto crea un objeto de contexto de conexión que especificará en cada sentencia de SQL que ejecute en la fuente de datos asociada. La sentencia de invocación del constructor debe tener uno de los formatos siguientes:

   ```
   clase-contexto-conexión objeto-contexto-conexión=
     new clase-contexto-conexión(String  url, boolean confirmación-automática);

   clase-contexto-conexión objeto-contexto-conexión =
     new clase-contexto-conexión(String  url, String usuario,
   String contraseña, boolean confirmación-automática);
   clase-contexto-conexión objeto-contexto-conexión =
     new clase-contexto-conexión(String url, Properties info,
   boolean confirmación-automática);
   ```

   Estos son los significados de los parámetros:

   url Es una cadena de caracteres que especifica el nombre de ubicación correspondiente a la fuente de datos. Este argumento tiene uno de los formatos especificados en el tema Conexión a una fuente de datos utilizando la interfaz DriverManager con el Controlador JDBC Universal. El formato depende del controlador JDBC que esté utilizando.

user **y** *password*
Especifique un ID de usuario y contraseña para la conexión con la fuente de datos, si son necesarios para la fuente de datos a la que se conecta.

info
Especifica un objeto de tipo `java.util.Properties` que contiene un conjunto de propiedades de controlador correspondientes a la conexión. Para el Controlador JDBC de DB2 de tipo 2 para Linux, UNIX® y Windows® (Controlador JDBC de DB2 de tipo 2), debe especificar solamente las propiedades `ususario` y `contraseña`. Para el Controlador JDBC universal de DB2, puede especificar cualquiera de las propiedades listadas en el tema Propiedades del Controlador JDBC de DB2 Universal.

confirmación-automática
Especifica si el gestor de bases de datos debe emitir una operación de confirmación después de cada sentencia. Los valores posibles son `true` o `false`. Si especifica `false`, necesitará realizar operaciones de confirmación explícitas.

El código siguiente utiliza la técnica de conexión 1 para crear una conexión con la ubicación NEWYORK. La conexión exige especificar un ID de usuario y contraseña, y no necesita confirmación automática. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
#sql context Ctx;       // Crear la clase de contexto de conexión Ctx  1
String userid="dbadm";       // Declarar variables para ID de usuario y contraseña
String password="dbadm";
String empname;              // Declarar una variable de lenguaje principal
...
try {                        // Cargar el controlador JDBC
  Class.forName("com.ibm.db2.jcc.DB2Driver");                     2
}
catch (ClassNotFoundException e) {
   e.printStackTrace();
}
Ctx myConnCtx=                                                        3
  new Ctx("jdbc:db2://sysmvs1.stl.ibm.com:5021/NEWYORK",
  userid,password,false);   // Crear el objeto de contexto de conexión myConnCtx
                            // para la conexión con NEWYORK
#sql [myConnCtx] {SELECT LASTNAME INTO :empname FROM EMPLOYEE
   WHERE EMPNO='000010'};
                     // Utilizar myConnCtx para ejecutar una sentencia de SQL
```

Figura 34. Uso de la técnica de conexión 1 para conectar con una fuente de datos

Técnica de conexión 2: esta técnica utiliza la interfaz `DriverManager` de JDBC para crear la conexión. Puede utilizar esta técnica con un nivel cualquiera del controlador JDBC.

1. Ejecute una cláusula de declaración de conexión de SQLJ.

   Esto es lo mismo que el paso 1 en la página 349 en la técnica de conexión 1.
2. Cargue el controlador.

   Esto es lo mismo que el paso 2 en la página 349 en la técnica de conexión 1.
3. Invoque el método `DriverManager.getConnection` de JDBC.

   Esto crea un objeto de conexión de JDBC para la conexión con la fuente de datos. Puede utilizar cualquiera de los formatos de `getConnection` especificados en el tema Conexión a una fuente de datos utilizando la interfaz DriverManager con el Controlador JDBC Universal.

Los significados de los parámetros *url*, *usuario* y *contraseña* son los mismos que para los parámetros del paso 3 en la página 349 en la técnica de conexión 1.

4. Invoque el constructor para la clase de contexto de conexión que creó en el paso 1 en la página 350.

   Esto crea un objeto de contexto de conexión que especificará en cada sentencia de SQL que ejecute en la fuente de datos asociada. La sentencia de invocación del constructor debe tener el formato siguiente:

   ```
   clase-contexto-conexión objeto-contexto-conexión=
     new clase-contexto-conexión(Connection objeto-conexión-JDBC);
   ```

   El parámetro *objeto-conexión-JDBC* es el objeto `Connection` que creó en el paso 3 en la página 350.

El código siguiente utiliza la técnica de conexión 2 para crear una conexión con la ubicación NEWYORK. La conexión exige especificar un ID de usuario y contraseña, y no necesita confirmación automática. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
#sql context Ctx;       // Crear la clase de contexto de conexión Ctx  1
String userid="dbadm";       // Declarar variables para ID de usuario y contraseña
String password="dbadm";
String empname;              // Declarar una variable de lenguaje principal
...
try {                        // Cargar el controlador JDBC
  Class.forName("com.ibm.db2.jcc.DB2Driver");                        2
}
catch (ClassNotFoundException e) {
   e.printStackTrace();
}
Connection jdbccon=                                                  3
  DriverManager.getConnection("jdbc:db2://sysmvs1.stl.ibm.com:5021/NEWYORK",
    userid,password);
                // Crear el objeto de conexión jdbccon de JDBC
jdbccon.setAutoCommit(false);  // No realizar confirmación automática  4
Ctx myConnCtx=new Ctx(jdbccon);                                      5
                           // Crear el objeto de contexto de conexión myConnCtx
                           // para la conexión con NEWYORK
#sql [myConnCtx] {SELECT LASTNAME INTO :empname FROM EMPLOYEE
   WHERE EMPNO='000010'};
                     // Utilizar myConnCtx para ejecutar una sentencia de SQL
```

Figura 35. Uso de la técnica de conexión 2 para conectar con una fuente de datos

Técnica de conexión 3: esta técnica utiliza la interfaz `DataSource` de JDBC para crear la conexión.

1. Ejecute una cláusula de declaración de conexión de SQLJ.

   Esto es lo mismo que el paso 1 en la página 349 en la técnica de conexión 1.

2. Si el administrador del sistema ha creado un objeto `DataSource` en un programa diferente:

   a. Obtenga el nombre lógico de la fuente de datos con la que necesita conectar.

   b. Cree un contexto para utilizarlo en el paso siguiente.

   c. En su programa de aplicación, utilice la interfaz JNDI (Java™ Naming and Directory Interface) para obtener el objeto `DataSource` que está asociado al nombre lógico de la fuente de datos.

En otro caso, cree un objeto DataSource y asígnele propiedades, tal como se muestra en el tema "Creación y utilización de una fuente de datos en la misma aplicación" en Conexión a una fuente de datos utilizando la interfaz DataSource.

3. Invoque el método DataSource.getConnection de JDBC.

   Esto crea un objeto de conexión de JDBC para la conexión con la fuente de datos. Puede utilizar uno de los formatos siguientes de getConnection:

   ```
   getConnection();
   getConnection(usuario, contraseña);
   ```

   Los significados de los parámetros *usuario* y *contraseña* son los mismos que para los parámetros del paso 3 en la página 349 en la técnica de conexión 1.

4. Si el valor por omisión para la confirmación automática no es apropiado, invoque el método Connection.setAutoCommit.

   De esta forma especifica si el gestor de bases de datos debe emitir una operación de confirmación después de cada sentencia. El formato de este método es:

   ```
   setAutoCommit(boolean confirmación-automática);
   ```

5. Invoque el constructor para la clase de contexto de conexión que creó en el paso 1 en la página 351.

   Esto crea un objeto de contexto de conexión que especificará en cada sentencia de SQL que ejecute en la fuente de datos asociada. La sentencia de invocación del constructor debe tener el formato siguiente:

   ```
   clase-contexto-conexión objeto-contexto-conexión=
     new clase-contexto-conexión(Connection objeto-conexión-JDBC);
   ```

   El parámetro *objeto-conexión-JDBC* es el objeto Connection que creó en el paso 3.

El código siguiente utiliza la técnica de conexión 3 para crear una conexión con una ubicación cuyo nombre lógico es jdbc/sampledb. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
import java.sql.*;
import javax.naming.*;
import javax.sql.*;
...
#sql context CtxSqlj;       // Crear la clase de contexto de conexión CtxSqlj  1
Context ctx=new InitialContext();                                             2b
DataSource ds=(DataSource)ctx.lookup("jdbc/sampledb");                        2c
Connection con=ds.getConnection();                                            3
String empname;              // Declarar una variable de lenguaje principal
...
con.setAutoCommit(false);  // No realizar confirmación automática   4
CtxSqlj myConnCtx=new CtxSqlj(con);                                           5
                           // Crear el objeto de contexto de conexión myConnCtx
#sql [myConnCtx] {SELECT LASTNAME INTO :empname FROM EMPLOYEE
   WHERE EMPNO='000010'};
                    // Utilizar myConnCtx para ejecutar una sentencia de SQL
```

Figura 36. Uso de la técnica de conexión 3 para conectar con una fuente de datos

Técnica de conexión 4 (Controlador JDBC universal de DB2 solamente): esta técnica utiliza la interfaz DataSource de JDBC para crear la conexión. Esta técnica **exige** que DataSource esté registrado en JNDI.

1. Consulte al administrador del sistema para conocer el nombre lógico de la fuente de datos con la que necesita conectar.
2. Ejecute una cláusula de declaración de conexión de SQLJ.

Para este tipo de conexión, la cláusula de declaración de conexión debe tener este formato:

```
#sql public static context nombre-clase-contexto
 with (dataSource="nombre-lógico");
```

El contexto de conexión debe estar declarado como public y static. *nombre-lógico* es el nombre de fuente de datos que obtuvo en el paso 1 en la página 352.

3. Invoque el constructor para la clase de contexto de conexión que creó en el paso 2 en la página 352.

   Esto crea un objeto de contexto de conexión que especificará en cada sentencia de SQL que ejecute en la fuente de datos asociada. La sentencia de invocación del constructor debe tener uno de los formatos siguientes:

```
clase-contexto-conexión objeto-contexto-conexión=
  new clase-contexto-conexión();

clase-contexto-conexión objeto-contexto-conexión =
  new clase-contexto-conexión (String usuario,
    String contraseña);
```

   Los significados de los parámetros *usuario* y *contraseña* son los mismos que para los parámetros del paso 3 en la página 349 en la técnica de conexión 1.

El código siguiente utiliza la técnica de conexión 4 para crear una conexión con una ubicación cuyo nombre lógico es `jdbc/sampledb`. La conexión exige utilizar un ID de usuario y contraseña.

```
#sql public static context Ctx
  with (dataSource="jdbc/sampledb");                              2
                            // Crear la clase de contexto de conexión Ctx
String userid="dbadm";      // Declarar variables para ID de usuario y contraseña
String password="dbadm";

String empname;              // Declarar una variable de lenguaje principal
...
Ctx myConnCtx=new Ctx(userid, password);                          3
                            // Crear el objeto de contexto de conexión myConnCtx
                            // para la conexión con jdbc/sampledb
#sql [myConnCtx] {SELECT LASTNAME INTO :empname FROM EMPLOYEE
   WHERE EMPNO='000010'};
                     // Utilizar myConnCtx para ejecutar una sentencia de SQL
```

Figura 37. Uso de la técnica de conexión 4 para conectar con una fuente de datos

Técnica de conexión 5: esta técnica utiliza la conexión por omisión para conectar con la fuente de datos. Utilice la conexión por omisión especificando sus sentencias de SQL sin un objeto de contexto de conexión. Cuando utiliza esta técnica, no es necesario cargar un controlador JDBC a menos que utilice explícitamente interfaces de JDBC en su programa. Por ejemplo:

```
#sql {SELECT LASTNAME INTO :empname FROM EMPLOYEE
   WHERE EMPNO='000010'};  // Utilizar conexión por omisión
                                    // para ejecutar una sentencia de SQL
```

Para crear un contexto de conexión por omisión, SQLJ busca `jdbc/defaultDataSource` en JNDI. Si no hay nada registrado, se emite una excepción de contexto nulo cuando SQLJ intenta acceder al contexto.

Conceptos relacionados:

- "Conexión de las aplicaciones JDBC a una fuente de datos" en la página 291

Tareas relacionadas:

- "Conexión a una fuente de datos utilizando la interfaz DriverManager con el Controlador JDBC de DB2 Universal" en la página 294
- "Conexión a una fuente de datos utilizando la interfaz DataSource" en la página 297

Información relacionada:

- "Propiedades del Controlador JDBC de DB2 Universal" en la página 400

Establecimiento del nivel de aislamiento para una transacción de SQLJ

Para establecer el nivel de aislamiento de una unidad de trabajo dentro de un programa SQLJ, utilice la cláusula SET TRANSACTION ISOLATION LEVEL. La Tabla 37 muestra los valores que puede especificar en la cláusula SET TRANSACTION ISOLATION LEVEL y los valores equivalentes para DB2®.

Tabla 37. Niveles de aislamiento equivalentes para SQLJ y DB2

Valor de SET TRANSACTION	Nivel de aislamiento de DB2
SERIALIZABLE	Lectura repetible
REPEATABLE READ	Estabilidad de lectura
READ COMMITTED	Estabilidad del cursor
READ UNCOMMITTED	Lectura no confirmada

El nivel de aislamiento afecta a la conexión JDBC subyacente así como a la conexión SQLJ. Puede cambiar el nivel de aislamiento solo al comienzo de una transacción.

Conceptos relacionados:

- "Niveles de aislamiento" en la publicación *Consulta de SQL, Volumen 1*

Confirmación o retrotracción de transacciones SQLJ

Si inhabilita la confirmación automática para una conexión SQLJ, será necesario realizar operaciones explícitas de confirmación o retrotracción. Para ello utiliza cláusulas de ejecución que contienen sentencias COMMIT o ROLLBACK de SQL:

```
#sql [myConnCtx] {COMMIT};
#sql [myConnCtx] {ROLLBACK};
```

Conceptos relacionados:

- "Puntos de rescate en aplicaciones SQLJ" en la página 354

Tareas relacionadas:

- "Conexión a una fuente de datos utilizando SQLJ" en la página 349

Puntos de rescate en aplicaciones SQLJ

Un punto de rescate de SQL representa el estado que tienen datos y esquemas en un momento determinado dentro de una unidad de trabajo. Existen sentencias de SQL para establecer un punto de rescate, liberar un punto de rescate y para restaurar datos y esquemas al estado representado por el punto de rescate.

Cuando se utiliza el Controlador JDBC universal de DB2, puede incluir en su programa SQLJ cualquier formato de la sentencia SAVEPOINT de SQL.

El ejemplo siguiente muestra cómo establecer un punto de rescate, retrotraer trabajos hasta un punto de rescate y liberar el punto de rescate.

```
#sql context Ctx;      // Crear la clase de contexto de conexión Ctx
String empNumVar;
int shoeSizeVar;
...
try {                         // Cargar el controlador JDBC
  Class.forName("com.ibm.db2.jcc.DB2Driver");
}
catch (ClassNotFoundException e) {
   e.printStackTrace();
}
Connection jdbccon=
  DriverManager.getConnection("jdbc:db2://sysmvs1.stl.ibm.com:5021/NEWYORK",
    userid,password);
               // Crear el objeto de conexión jdbccon de JDBC
jdbccon.setAutoCommit(false);  // No realizar confirmación automática
Ctx ctxt=new Ctx(jdbccon);
                               // Crear el objeto de contexto de conexión myConnCtx
                               // para la conexión con NEWYORK
#sql [ctxt] {CREATE DISTINCT TYPE SHOESIZE AS INTEGER WITH COMPARISONS};
                              // Crear un tipo diferenciado
#sql [ctxt] {COMMIT};
                              // Confirmar la creación
#sql [ctxt]
  {CREATE TABLE EMP_SHOE (EMPNO CHAR(6), EMP_SHOE_SIZE SHOESIZE)};
                             // Crear tabla con tipo diferenciado
#sql [ctxt] {COMMIT};
                              // Confirmar la creación
#sql [ctxt]
  {INSERT INTO EMP_SHOE VALUES ('000010', 6)};
                               // Insertar una fila
#sql [ctxt]
  {SAVEPOINT SVPT1 ON ROLLBACK RETAIN CURSORS};
                               // Crear un punto de rescate
...
#sql [ctxt]
  {INSERT INTO EMP_SHOE VALUES ('000020', 10)};
                                // Insertar otra fila
#sql [ctxt] {ROLLBACK TO SAVEPOINT SVPT1};
                             // Retrotraer trabajo hasta el punto
                             // después de la primera inserción
...
#sql [ctxt] {RELEASE SAVEPOINT SVPT1};
                               // Liberar el punto de rescate
ctx.close();                   // Cerrar el contexto de conexión
```

Figura 38. Establecimiento, retrotracción y liberación de un punto de rescate en una aplicación SQLJ

Tareas relacionadas:

- "Confirmación o retrotracción de transacciones SQLJ" en la página 354

Información relacionada:

- "Sentencia ROLLBACK" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia RELEASE SAVEPOINT" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia SAVEPOINT" en la publicación *Consulta de SQL, Volumen 2*

Cierre de la conexión con una fuente de datos en una aplicación SQLJ

Una vez finalizada una conexión con una fuente de datos, es necesario que cierre la conexión con la fuente de datos. De esta forma se liberan inmediatamente los recursos del objeto de contexto de conexión para DB2® y SQLJ.

Para cerrar la conexión con la fuente de datos, utilice el método `ConnectionContext.close()`. Esto cierra el contexto de conexión así como la conexión con la fuente de datos. Por ejemplo:

```
...
ctx = new EzSqljctx(con0);      // Crear objeto de contexto de conexión
                                // desde la conexión con0 de JDBC
...                                 // Ejecutar diversas operaciones de SQL
 EzSqljctx.close();             // Cerrar el contexto de conexión y
                                // la conexión con la fuente de datos
```

Tareas relacionadas:

- "Conexión a una fuente de datos utilizando SQLJ" en la página 349

Sentencias de SQL en una aplicación SQLJ

En un programa SQL normal, puede ejecutar sentencias de SQL para crear tablas, insertar, actualizar, suprimir o recuperar datos de tablas, invocar procedimientos almacenados, o confirmar o retrotraer transacciones. En un programa SQLJ, puede también ejecutar esas sentencias, dentro de *cláusulas ejecutables* de SQLJ. Una cláusula ejecutable puede tener uno de los formatos generales siguientes:

```
#sql [contexto-conexión] {sentencia-sql};
#sql [contexto-conexión,contexto-ejecución] {sentencia-sql};
#sql [contexto-ejecución] {sentencia-sql};
```

En una cláusula ejecutable, debe especificar *siempre* un contexto de conexión explícito, con una sola excepción: no es necesario especificar un contexto de conexión explícito para una sentencia FETCH. Solo debe incluir un contexto de ejecución para casos específicos. Consulte el tema Control de la ejecución de sentencias de SQL en SQLJ para obtener información sobre cuándo es necesario un contexto de ejecución.

Conceptos relacionados:

- "Comentarios en una aplicación SQLJ" en la página 348
- "Uso de SQLJ y JDBC en la misma aplicación" en la página 373
- "Datos LOB en aplicaciones JDBC con el Controlador JDBC de DB2 Universal" en la página 376
- "Recuperación de varios conjuntos de resultados de un procedimiento almacenado en una aplicación SQLJ" en la página 382
- "Recuperación de datos de tablas DB2 por una aplicación SQLJ" en la página 357

Tareas relacionadas:

- "Realización de actualizaciones por lotes en aplicaciones SQLJ" en la página 383
- "Invocación de procedimientos almacenados en una aplicación SQLJ" en la página 370
- "Confirmación o retrotracción de transacciones SQLJ" en la página 354

- "Creación y modificación de objetos DB2 en una aplicación SQLJ" en la página 357
- "Manejo de errores de SQL en una aplicación SQLJ" en la página 371
- "Establecimiento del nivel de aislamiento para una transacción de SQLJ" en la página 354
- "Utilización de un iterador de nombre en una aplicación SQLJ" en la página 358
- "Utilización de un iterador de posición en una aplicación SQLJ" en la página 361
- "Ejecución de operaciones UPDATE y DELETE de posición en una aplicación SQLJ" en la página 363
- "Utilización de iteradores desplazables en una aplicación SQLJ" en la página 389
- "Manejo de avisos de SQL en una aplicación SQLJ" en la página 372
- "Control de la ejecución de sentencias de SQL en SQLJ" en la página 381

Información relacionada:

- "Cláusula ejecutable de SQLJ" en la página 433

Creación y modificación de objetos DB2 en una aplicación SQLJ

Utilice cláusulas ejecutables de SQLJ para realizar lo siguiente:

- Ejecutar sentencias de definición de datos (CREATE, ALTER, DROP, GRANT, REVOKE)
- Ejecutar sentencias INSERT, y sentencias UPDATE y DELETE de búsqueda

Por ejemplo, las sentencias ejecutables siguientes muestran una sentencia INSERT, una sentencia UPDATE de búsqueda y una sentencia DELETE de búsqueda:

```
#sql [myConnCtx] {INSERT INTO DEPARTMENT VALUES
  ("X00","Operations 2","000030","E01",NULL)};
#sql [myConnCtx] {UPDATE DEPARTMENT
  SET MGRNO="000090" WHERE MGRNO="000030"};
#sql [myConnCtx] {DELETE FROM DEPARTMENT
  WHERE DEPTNO="X00"};
```

Para obtener información sobre las sentencias UPADATE y DELETE posicionadas, consulte el tema Ejecución de operaciones UPDATE y DELETE posicionadas en una aplicación SQL.

Tareas relacionadas:

- "Ejecución de operaciones UPDATE y DELETE de posición en una aplicación SQLJ" en la página 363

Recuperación de datos de tablas DB2 por una aplicación SQLJ

Al igual que ocurre en aplicaciones DB2® escritas en otros lenguajes de programación, si desea recuperar una fila individual de una tabla DB2 en una aplicación SQLJ, puede escribir una sentencia SELECT INTO con una cláusula WHERE que define una tabla de resultados que contiene solo esa fila:

```
#sql [myConnCtx] {SELECT DEPTNO INTO :hvdeptno FROM DEPARTMENT WHERE
     DEPTNAME="OPERATIONS"};
```

Sin embargo, las mayoría de las sentencias SELECT que utiliza producen tablas de resultados que contienen muchas filas. En las aplicaciones DB2 escritas en otros

lenguajes, se utiliza un cursor para seleccionar filas individuales de la tabla de resultados. Ese cursor puede ser no desplazable, lo que significa que cuando lo utiliza para recuperar filas, desplaza el cursor secuencialmente desde el principio de la tabla de resultados hasta el final. Como alternativa, el cursor puede ser desplazable, lo que significa que cuando lo utiliza para recuperar filas, puede desplazar el cursor hacia delante y atrás o situarlo en una fila cualquiera de la tabla de resultados.

En SQLJ, el equivalente de un cursor es un *iterador de conjunto de resultados*. Al igual que un cursor, un iterador de conjunto de resultados puede ser desplazable o no desplazable. El presente tema describe cómo utilizar iteradores no desplazables. Para obtener información sobre el uso de iteradores desplazables, consulte el tema Utilización de iteradores desplazables en una aplicación SQLJ.

Un iterador de conjunto de resultados es un objeto Java™ que se utiliza para recuperar filas de una tabla de resultados. A diferencia de un cursor, un iterador de conjunto de resultados se puede pasar como parámetro a un método.

Estos son los pasos básicos para utilizar un iterador de conjunto de resultados:
1. Declare el iterador, con lo que se creará una clase de iterador
2. Defina una instancia de la clase de iterador.
3. Asigne la tabla de resultados de una sentencia SELECT a una instancia del iterador.
4. Recupere filas.
5. Cierre el iterador.

Existen dos tipos de iteradores: *iteradores de posición* e *iteradores de nombre*. Los iteradores de posición amplían la interfaz `sqlj.runtime.PositionedIterator`. Los iteradores de posición identifican las columnas de una tabla de resultados por la posición que ocupan en la tabla de resultados. Los iteradores de nombre amplían la interfaz `sqlj.runtime.NamedIterator`. Los iteradores de nombre identifican las columnas de la tabla de resultados por su nombre.

Tareas relacionadas:

- “Utilización de un iterador de nombre en una aplicación SQLJ” en la página 358
- “Utilización de un iterador de posición en una aplicación SQLJ” en la página 361
- “Ejecución de operaciones UPDATE y DELETE de posición en una aplicación SQLJ” en la página 363

Información relacionada:

- “Cláusula de declaración de iterador de SQLJ” en la página 431

Utilización de un iterador de nombre en una aplicación SQLJ

Estos son los pasos para utilizar un iterador de nombre:

1. Declare el iterador.

   Utilice una *cláusula de declaración de iterador* para declarar un iterador de conjunto de resultados. Esto hace que se cree una clase de iterador que tiene el mismo nombre que el iterador. Para un iterador de nombre, la cláusula de declaración de iterador especifica la información siguiente:

   - El nombre del iterador
   - Una lista de nombres de columnas y tipos de datos Java™

- Información para una declaración de clase Java, tal como la indicación de si el iterador es público (`public`) o estático (`static`)
- Un conjunto de atributos, tales como si el iterador se puede retener o si sus columnas se pueden actualizar.

Cuando declara un iterador de nombre para una consulta, especifica nombres para cada columna del iterador. Esos nombres deben coincidir con los nombres de las columnas de la tabla de resultados de la consulta, sin que sea necesario que coincidan en el uso de las letras mayúsculas y minúsculas. La clase de iterador de nombre generada por la cláusula de declaración de iterador contiene *métodos accesores*. Existe un método accesor para cada columna del iterador. Cada método accesor tiene el mismo nombre que la columna correspondiente del iterador. Los métodos accesores se utilizan para recuperar datos contenidos en columnas de la tabla de resultados.

Es necesario que en los iteradores especifique tipos de datos Java que se correspondan estrechamente con los correspondientes tipos de datos de columnas DB2®. Consulte el tema Tipos de datos de Java, JDBC y SQL para obtener una lista de los tipos de datos compatibles de Java y DB2.

Puede declarar un iterador de varias maneras. Sin embargo, debido a que cada iterador está asociado a una clase Java, es necesario que cuando declare un iterador, la clase asociada cumpla las reglas de Java. Por ejemplo, los iteradores que contienen una *cláusula with* se deben declarar como `public`. Por tanto, si es necesario que un iterador sea `public`, se puede declarar solo donde esté permitida una clase `public`. La lista siguiente describe algunos métodos alternativos para declarar un iterador:

- Como `public`, en un archivo fuente separado

  Este método le permite utilizar la declaración de iterador en otros módulos de código, y proporciona un iterador que es efectivo para todas las aplicaciones SQLJ. Además, no existen las cuestiones derivadas de tener otras clases de nivel superior o clases `public` en el mismo archivo fuente.
- Como clase de nivel superior en un archivo fuente que contiene otras definiciones de clases de nivel superior

  Java permite una sola clase pública, de nivel superior, en un módulo de código. Por tanto, si necesita declarar el iterador como público, tal como cuando el iterador incluye una cláusula with, ninguna otra clase contenida en el módulo de código puede estar declarada como pública.
- Como clase estática anidada dentro de otra clase

  Esta alternativa le permite combinar la declaración de iterador con otras declaraciones de clases en el mismo archivo fuente, declarar el iterador y otras clases como públicos y hacer que la clase de iterador sea visible para otros módulos de código o paquetes. Si embargo, si especifica el iterador desde fuera de la clase anidadora debe calificar por completo el nombre de iterador con el nombre de la clase anidadora.
- Como clase interna dentro de otra clase

  Cuando declara un iterador de esta manera, puede crear una instancia del iterador solo dentro de una instancia de la clase anidadora. Sin embargo, puede declarar como públicos el iterador y otras clases contenidas en el archivo.

  No puede convertir un `conjunto de resultados` de JDBC en un iterador si el iterador está declarado como clase interna. Esta restricción no es aplicable si el iterador está declarado como clase anidada estática. Consulte el tema Utilización de SQLJ y JDBC en la misma aplicación para obtener más información sobre la conversión de un `conjunto de resultados` en un iterador.

2. Cree una instancia de la clase de iterador.

   Debe declarar un objeto de la clase de iterador de nombre para recuperar filas de una tabla de resultados.

3. Asigne la tabla de resultados de una sentencia SELECT a una instancia del iterador.

   Para asignar la tabla de resultados de una sentencia SELECT a un iterador, utilice una *cláusula de asignación* de SQLJ. Este es el formato de la cláusula de asignación para un iterador de nombre:

   ```
   #sql cláusula-contexto objeto-iterador={sentencia-select};
   ```

   Consulte los temas Cláusula de asignación de SQLJ y Cláusula de contexto de SQLJ para obtener más información.

4. Recupere filas.

   Para ello invoque métodos accesores en un bucle. Los métodos accesores tienen los mismos nombres que las columnas correspondientes del iterador, y carecen de parámetros. Un método accesor devuelve el valor de la columna correspondiente de la fila actual de la tabla de resultados. Utilice el método `NamedIterator.next()` para avanzar el cursor por la tabla de resultados.

   Para determinar si ha recuperado todas las filas, examine el valor devuelto al invocar el método `next`. `next` devuelve el valor `booleano false` si no existe ninguna fila siguiente.

5. Cierre el iterador.

   Para ello utilice el método `NamedIterator.close`.

El código siguiente muestra cómo declarar y utilizar un iterador de nombre. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
#sql  iterator ByName(String LastName, Date HireDate);        1
                       // Declarar iterador de nombre ByName
{
  ByName nameiter;    // Declarar objeto de la clase ByName        2
  #sql [ctxt]
 nameiter={SELECT LASTNAME, HIREDATE FROM EMPLOYEE};              3
                            // Asignar tabla de resultados de SELECT
                            // al objeto de iterador nameiter
  while (nameiter.next())   // Mover el iterador por la tabla de   4
                            // resultados y determinar si se han
                            // recuperado todas las filas
  {
    System.out.println( nameiter.LastName() + " was hired on "
      + nameiter.HireDate());   // Utilizar los métodos accesores
                                // LastName y HireDate para recuperar
                                // valores de columnas
  }
  nameiter.close();             // Cerrar el iterador             5
}
```

Figura 39. Utilización de un iterador de nombre

Conceptos relacionados:

- "Uso de SQLJ y JDBC en la misma aplicación" en la página 373

Tareas relacionadas:

- "Utilización de un iterador de posición en una aplicación SQLJ" en la página 361

- "Ejecución de operaciones UPDATE y DELETE de posición en una aplicación SQLJ" en la página 363

Información relacionada:

- "Tipos de datos de Java, JDBC y SQL" en la página 395
- "Cláusula de asignación de SQLJ" en la página 437
- "Cláusula context de SQLJ" en la página 434

Utilización de un iterador de posición en una aplicación SQLJ

Estos son los pasos para utilizar un iterador de posición:

1. Declare el iterador.

   Utilice una *cláusula de declaración de iterador* para declarar un iterador de conjunto de resultados. Esto hace que se cree una clase de iterador que tiene el mismo nombre y atributos que el iterador. Para un iterador de posición, la cláusula de declaración de iterador especifica la información siguiente:

   - El nombre del iterador
   - Una lista de tipos de datos Java™
   - Información para una declaración de clase Java, tal como la indicación de si el iterador es público (`public`) o estático (`static`)
   - Un conjunto de atributos, tales como si el iterador se puede retener o si sus columnas se pueden actualizar.

   Las declaraciones de tipos de datos representan columnas de la tabla de resultados y se especifican como columnas del iterador del conjunto de resultados. Las columnas del iterador del conjunto de resultados se corresponden con las columnas de la tabla de resultados, en el orden de izquierda a derecha. Por ejemplo, si una cláusula de declaración de iterador tiene dos declaraciones de tipos de datos, la primera declaración corresponde a la primera columna de la tabla de resultados, y la segunda declaración corresponde a la segunda columna de la tabla de resultados.

   Es necesario que en los iteradores especifique tipos de datos Java que se correspondan estrechamente con los correspondientes tipos de datos de columnas DB2®. Consulte el tema Tipos de datos de Java, JDBC y SQL para obtener una lista de los tipos de datos compatibles de Java y DB2.

   Puede declarar un iterador de varias maneras. Sin embargo, debido a que cada iterador está asociado a una clase Java, cuando declare un iterador, la clase subyacente debe cumplir las reglas de Java. Por ejemplo, los iteradores que contienen una *cláusula with* se deben declarar como `public`. Por tanto, si es necesario que un iterador sea `public`, se puede declarar solo donde esté permitida una clase `public`. La lista siguiente describe algunos métodos alternativos para declarar un iterador:

   - Como `public`, en un archivo fuente separado

     Este es el método más versátil de declarar un iterador. Este método le permite utilizar la declaración de iterador en otros módulos de código, y proporciona un iterador que es efectivo para todas las aplicaciones SQLJ. Además, no existen las cuestiones derivadas de tener otras clases de nivel superior o clases `public` en el mismo archivo fuente.
   - Como clase de nivel superior en un archivo fuente que contiene otras definiciones de clases de nivel superior

     Java permite una sola clase pública, de nivel superior, en un módulo de código. Por tanto, si necesita declarar el iterador como público, tal como

cuando el iterador incluye una cláusula with, ninguna otra clase contenida en el módulo de código puede estar declarada como pública.

- Como clase estática anidada dentro de otra clase

  Esta alternativa le permite combinar la declaración de iterador con otras declaraciones de clases en el mismo archivo fuente, declarar el iterador y otras clases como públicos y hacer que la clase de iterador sea visible para otros módulos de código o paquetes. Si embargo, si especifica el iterador desde fuera de la clase anidadora debe calificar por completo el nombre de iterador con el nombre de la clase anidadora.

- Como clase interna dentro de otra clase

  Cuando declara un iterador de esta manera, puede crear una instancia del iterador solo dentro de una instancia de la clase anidadora. Sin embargo, puede declarar como públicos el iterador y otras clases contenidas en el archivo.

  No puede convertir un `conjunto de resultados` de JDBC en un iterador si el iterador está declarado como clase interna. Esta restricción no es aplicable si el iterador está declarado como clase anidada estática. Consulte el tema Utilización de SQLJ y JDBC en la misma aplicación para obtener más información sobre la conversión de un `conjunto de resultados` en un iterador.

2. Cree una instancia de la clase de iterador.

   Debe declarar un objeto de la clase de iterador de posición para recuperar filas de una tabla de resultados.

3. Asigne la tabla de resultados de una sentencia SELECT a una instancia del iterador.

   Para asignar la tabla de resultados de una sentencia SELECT a un iterador, utilice una *cláusula de asignación* de SQLJ. Este es el formato de la cláusula de asignación para un iterador de posición:

   ```
   #sql cláusula-contexto objeto-iterador={sentencia-select};
   ```

4. Recupere filas.

   Para ello ejecute en bucle sentencias FETCH en cláusulas ejecutables. Las sentencias FETCH son iguales a las utilizadas en otros lenguajes de programación.

   Para determinar si ha recuperado todas las filas, invoque el método `PositionedIterator.endFetch` después de cada FETCH. `endFetch` devuelve el valor `booleano true` si no hay más filas para recuperar mediante FETCH.

5. Cierre el iterador.

   Para ello utilice el método `PositionedIterator.close`.

El código siguiente muestra cómo declarar y utilizar un iterador de posición. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
#sql iterator ByPos(String,Date); // Declarar iterador de posición ByPos 1
{
  ByPos positer;             // Declarar objeto de la clase ByPos      2
  String name = null;        // Declarar variables de lenguaje principal
  Date hrdate;
  #sql [ctxt] positer =
    {SELECT LASTNAME, HIREDATE FROM EMPLOYEE};                          3
                         // Asignar tabla de resultados de SELECT
                         // al objeto de iterador positer
  #sql {FETCH :positer INTO :name, :hrdate };                           4
                              // Recuperar primera fila
  while (!positer.endFetch())   // Determinar si FETCH ha devuelto una fila
  { System.out.println(name + " was hired in " +
      hrdate);
    #sql {FETCH :positer INTO :name, :hrdate };
                              // Obtener la fila siguiente
  }
  positer.close();             // Cerrar el iterador              5
}
```

Figura 40. Utilización de un iterador de posición

Conceptos relacionados:

- "Uso de SQLJ y JDBC en la misma aplicación" en la página 373
- "Recuperación de datos de tablas DB2 por una aplicación SQLJ" en la página 357

Tareas relacionadas:

- "Utilización de un iterador de nombre en una aplicación SQLJ" en la página 358

Información relacionada:

- "Tipos de datos de Java, JDBC y SQL" en la página 395

Ejecución de operaciones UPDATE y DELETE de posición en una aplicación SQLJ

Al igual que ocurre en las aplicaciones DB2® escritas en otros lenguajes, la ejecución de operaciones UPDATE y DELETE de posición es un caso ampliado de la recuperación de filas en una tabla de resultados. Los pasos básicos son:

1. Declare el iterador.

   El iterador puede ser de posición o de nombre. Para las operaciones UPDATE o DELETE de posición, el iterador se debe declarar como actualizable. Para ello, la declaración debe incluir las cláusulas siguientes:

   implements sqlj.runtime.ForUpdate
   Esta cláusula hace que la clase de iterador creada incluya métodos para utilizar iteradores actualizables. Esta cláusula es necesaria para los programas con operaciones UPDATE o DELETE de posición.

   with (updateColumns=*"lista-columnas"***)**
   Esta cláusula especifica una lista de las columnas, separadas por comas, de la tabla de resultados que serán actualizadas por el iterador. Esta cláusula es opcional.

   Debe declarar el iterador como `public`, por lo que es necesario que siga las reglas para declarar y utilizar iteradores `public` en el mismo archivo o archivos diferentes.

Si declara el iterador en un archivo separado, cualquier archivo fuente SQLJ que pueda acceder al iterador e importe la clase generada puede recuperar datos y ejecutar sentencias UPDATE o DELETE de posición utilizando el iterador. El ID de autorización utilizado para ejecutar la sentencia UPDATE o DELETE de posición depende de si la sentencia se ejecuta de forma estática o dinámica. Si la sentencia se ejecuta de forma estática, el ID de autorización es el propietario del plan o paquete DB2 donde está incluida la sentencia. Si la sentencia se ejecuta de forma dinámica, el ID de autorización está determinado por la acción de DYNAMICRULES que esté en vigor. Para el Controlador JDBC universal de DB2, la acción es siempre DYNAMICRULES BIND.

2. Inhabilite la modalidad de confirmación automática (autocommit) para la conexión.

   Cuando la modalidad de confirmación automática está habilitada, se ejecuta una operación COMMIT cada vez que se ejecuta una sentencia UPDATE de posición, lo que provoca la destrucción del iterador a menos que el iterador tenga el atributo `with (holdability=true)`. Por tanto, es necesario desactivar la confirmación automática para evitar que se ejecuten operaciones COMMIT hasta que haya terminado de utilizar el iterador. Si desea que se ejecute una operación COMMIT después de cada actualización, una forma alternativa de impedir la destrucción del iterador después de cada operación COMMIT es declarar el iterador con el atributo `with (holdability=true)`.

3. Cree una instancia de la clase de iterador.

   Este paso es el mismo que para un iterador no actualizable.

4. Asigne la tabla de resultados de una sentencia SELECT a una instancia del iterador.

   Este paso es el mismo que para un iterador no actualizable. La sentencia SELECT no debe incluir una cláusula FOR UPDATE.

5. Recupere y actualice filas.

   Para un iterador de posición, ejecute las acciones siguientes en bucle:

   a. Ejecute una sentencia FETCH en una cláusula ejecutable para obtener la fila actual.
   b. Invoque el método `PositionedIterator.endFetch` para determinar si el iterador está apuntando a una fila de la tabla de resultados.
   c. Si el iterador está apuntando a una fila de la tabla de resultados, ejecute una sentencia UPDATE... WHERE CURRENT OF *:objeto-iterador* de SQL en una cláusula ejecutable para actualizar las columnas de la fila actual. Ejecute una sentencia DELETE... WHERE CURRENT OF *:objeto-iterador* de SQL en una cláusula ejecutable para suprimir la fila actual.

   Para un iterador de nombre, ejecute las acciones siguientes en bucle:

   a. Invoque el método `next` para avanzar el iterador.
   b. Determine si el iterador está apuntando a una fila de la tabla de resultados; para ello compruebe si `next` devuelve el valor `true`.
   c. Ejecute una sentencia UPDATE... WHERE CURRENT OF *:objeto-iterador* de SQL en una cláusula ejecutable para actualizar las columnas de la fila actual. Ejecute una sentencia DELETE... WHERE CURRENT OF *:objeto-iterador* de SQL en una cláusula ejecutable para suprimir la fila actual.

6. Cierre el iterador.

   Para ello utilice el método `close`.

El código siguiente muestra cómo declarar un iterador de posición y utilizarlo para operaciones UPDATE de posición. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

En primer lugar, en un archivo individual, declare un iterador de posición UpdByPos, y especifique que desea utilizar el iterador para actualizar la columna SALARY:

```
import java.math.*;  // Importar esta clase para el tipo de datos BigDecimal
#sql public iterator UpdByPos implements sqlj.runtime.ForUpdate      1
  with(updateColumns="SALARY") (String, BigDecimal);
```

Figura 41. Declaración de un iterador de posición para una operación UPDATE de posición

A continuación, en otro archivo, utilice UpdByPos para un UPDATE de posición, tal como se muestra en el fragmento de código siguiente:

```
import sqlj.runtime.*;  // Importar archivos para las API de SQLJ y JDBC
import java.sql.*;
import java.math.*;  // Importar esta clase para el tipo de datos BigDecimal
import UpdByPos;     // Importar la clase de iterador generada que
                     // fue creada por la cláusula de declaración
                     // de iterador para UpdByName en otro archivo
#sql context HSCtx;  // Crear la clase de contexto de conexión HSCtx
public static void main (String args[])
{
try {
    Class.forName("com.ibm.db2.jcc.DB2Driver");
  }
  catch (ClassNotFoundException e) {
    e.printStackTrace();
          }
  Connection HSjdbccon=
  DriverManager.getConnection("jdbc:db2:SANJOSE");
                     // Crear un objeto de conexión de JDBC
  HSjdbccon.setAutoCommit(false);
                     // Desactivar la confirmación automática        2
                     // para impedir que destruya el cursor
                     // entre actualizaciones
  HSCtx myConnCtx=new HSCtx(HSjdbccon);
                     // Crear un objeto de contexto de conexión
  UpdByPos upditer;  // Declarar un objeto de iterador de clase UpdByPos class  3
  String enum;       // Declarar variable de lenguaje principal para contener
  BigDecimal sal;    // los valores de las columnas EMPNO y SALARY
  #sql [myConnCtx]
    upditer = {SELECT EMPNO, SALARY FROM EMPLOYEE                    4
      WHERE WORKDEPT='D11'};
                     // Asignar tabla de resultados a objeto de iterador
  #sql {FETCH :upditer INTO :enum,:sal};                             5a
                           // Avanzar cursor hasta la fila siguiente
  while (!upditer.endFetch())                                        5b
                           // Determinar si iterador apunta a una fila
  {
    #sql [myConnCtx] {UPDATE EMPLOYEE SET SALARY=SALARY*1.05
      WHERE CURRENT OF :upditer};                                    5c
                           // Ejecutar actualización de posición
    System.out.println("Updating row for " + enum);
    #sql {FETCH :upditer INTO :enum,:sal};
                           // Avanzar cursor hasta la fila siguiente
  }
  upditer.close();         // Cerrar el iterador                    6
  #sql [myConnCtx] {COMMIT};
                           // Confirmar los cambios
  myConnCtx.close();       // Cerrar el contexto de conexión
}
```

Figura 42. Ejecución de un UPDATE de posición con un iterador de posición

El código siguiente muestra cómo declarar un iterador de nombre y utilizarlo para operaciones UPDATE de posición. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

En primer lugar, en un archivo individual, declare el iterador de nombre `UpdByName`, y especifique que desea utilizar el iterador para actualizar la columna `SALARY`:

```
import java.math.*;  // Importar esta clase para el tipo de datos BigDecimal
#sql public iterator UpdByName implements sqlj.runtime.ForUpdate     1
  with(updateColumns="SALARY") (String EmpNo, BigDecimal Salary);
```

Figura 43. Declaración de un iterador de nombre para una operación UPDATE de posición

A continuación, en otro archivo, utilice UpdByName para un UPDATE de posición, tal como se muestra en el fragmento de código siguiente:

```
import sqlj.runtime.*;  // Importar archivos para las API de SQLJ y JDBC
import java.sql.*;
import java.math.*;  // Importar esta clase para el tipo de datos BigDecimal
import UpdByName;    // Importar la clase de iterador generada que
                     // fue creada por la cláusula de declaración
                     // de iterador para UpdByName en otro archivo
#sql context HSCtx;  // Crear la clase de contexto de conexión HSCtx
public static void main (String args[])
{
try {
    Class.forName("com.ibm.db2.jcc.DB2Driver");
  }
  catch (ClassNotFoundException e) {
    e.printStackTrace();
          }
  Connection HSjdbccon=
  DriverManager.getConnection("jdbc:db2:SANJOSE");
                     // Crear un objeto de conexión de JDBC
  HSjdbccon.setAutoCommit(false);
                     // Desactivar la confirmación automática        2
                     // para impedir que destruya el cursor
                     // entre actualizaciones
  HSCtx myConnCtx=new HSCtx(HSjdbccon);
                     // Crear un objeto de contexto de conexión
  UpdByName upditer;                                              3
                    // Declarar objeto de iterador de clase UpdByName
  String enum;      // Declarar variable de lenguaje principal
                    // para contener valores de columnas
  #sql [myConnCtx]
    upditer = {SELECT EMPNO, SALARY FROM EMPLOYEE                 4
      WHERE WORKDEPT='D11'};
                     // Asignar tabla de resultados a objeto de iterador
  while (upditer.next())                                          5a, 5b
                             // Avanzar cursor hasta la fila siguiente
                             // Determinar si iterador apunta a una fila
  {
    enum = upditer.EmpNo();  // Obtener número de empleado en la fila actual
    #sql [myConnCtx]
      {UPDATE EMPLOYEE SET SALARY=SALARY*1.05
        WHERE CURRENT OF :upditer};                               5c
                             // Ejecutar actualización de posición
    System.out.println("Updating row for " + enum);
  }
  upditer.close();           // Cerrar el iterador                6
  #sql [myConnCtx] {COMMIT};
                             // Confirmar los cambios
  myConnCtx.close();         // Cerrar el contexto de conexión
}
```

Figura 44. Ejecución de un UPDATE de posición con un iterador de nombre

Conceptos relacionados:

- "Recuperación de datos de tablas DB2 por una aplicación SQLJ" en la página 357

- "Uso de iteradores pasados como variables en operaciones UPDATE o DELETE de posición de una aplicación SQLJ" en la página 387

Tareas relacionadas:

- "Conexión a una fuente de datos utilizando SQLJ" en la página 349

Varios iteradores abiertos para una misma sentencia de SQL en una aplicación SQLJ

Si está utilizando el Controlador JDBC universal de DB2, y su aplicación se conecta a un servidor DB2 UDB para z/OS® Versión 8, o a un servidor DB2 UDB para Linux, UNIX® y Windows® con el FixPak 4 o posterior, puede tener abiertos a la vez varios iteradores para una misma sentencia de SQL en una aplicación SQLJ. Esta característica le permite efectuar una operación en una tabla utilizando un iterador mientras ejecuta una operación diferente en la misma tabla utilizando otro iterador.

Cuando utiliza simultáneamente varios iteradores abiertos en una aplicación, es conveniente que cierre los iteradores que ya no necesite, a fin de evitar un consumo excesivo de espacio de almacenamiento dinámico de Java™.

Los ejemplos siguientes muestran la ejecución de unas mismas operaciones en una tabla con y sin operadores abiertos simultáneamente para una misma sentencia de SQL. Estos ejemplos utilizan la siguiente declaración de iterador:

```
import java.math.*;
#sql public iterator  MultiIter(String EmpNo, BigDecimal Salary);
```

Si no se utiliza la capacidad de tener abiertos a la vez varios iteradores para una misma sentencia de SQL, si desea seleccionar valores de empleado y salario para un número de empleado determinado, debe definir una sentencia de SQL diferente para cada número de empleado, tal como muestra la Figura 45.

```
MultiIter iter1 = null;       // Instancia de iterador para
                                // recuperar datos para el primer empleado
String EmpNo1 = "000100";       // Número de empleado del primer empleado
#sql [ctx] iter2 =
  {SELECT EMPNO, SALARY FROM EMPLOYEE WHERE EMPNO = :EmpNo1};
                                // Asignar tabla de resultados a primer iterador
MultiIter iter2 = null;       // Instancia de iterador para recuperar
                                // datos para el segundo empleado
String EmpNo2 = "000200";     // Número de empleado del segundo empleado
#sql [ctx] iter2 =
  {SELECT EMPNO, SALARY FROM EMPLOYEE WHERE EMPNO = :EmpNo2};
                                // Asignar tabla de resultados a segundo iterador
  // Procesar con iter1
 // Procesar con iter2
iter1.close();                // Cerrar los iteradores
iter2.close();
```

Figura 45. Ejemplo de operaciones de tabla simultáneas utilizando iteradores con sentencias de SQL diferentes

La Figura 46 en la página 369 muestra cómo ejecutar las mismas operaciones cuando existe la capacidad de tener abiertos a la vez varios iteradores para una misma sentencia de SQL.

```
...
MultiIter iter1 = openIter("000100"); // Invocar openIter para asignar
                                      // la tabla de resultados
                                      // (para el empleado 100) al primer
                                      // iterador
MultiIter iter2 = openIter("000200"); // Invocar openIter para asignar la
                                      // tabla de resultados al segundo
                                      // iterador.
                                      // iter1 permanece abierto cuando
                                      // se abre iter2
// Procesar con iter1
// Procesar con iter2
...
iter1.close();                        // Cerrar los iteradores
iter2.close();
...
public MultiIter openIter(String EmpNo)
                               // Método para asignar una tabla de
                               // resultados a una instancia de iterador
{
  MultiIter iter;
  #sql [ctxt] iter =
    {SELECT EMPNO, SALARY FROM EMPLOYEE WHERE EMPNO = :EmpNo};
  return iter;                 // El método devuelve una instancia de iterador
}
```

Figura 46. Ejemplo de operaciones de tabla simultáneas utilizando iteradores con la misma sentencia de SQL

Conceptos relacionados:

- "Recuperación de datos de tablas DB2 por una aplicación SQLJ" en la página 357

Uso de varias instancias abiertas de un iterador en una aplicación SQLJ

Pueden estar abiertas simultáneamente varias instancias de un iterador en una misma aplicación SQLJ. Una utilización de esta capacidad es abrir varias instancias de un iterador que hace uso de expresiones de lenguaje principal. Cada instancia puede utilizar un conjunto diferente de valores para una expresión de lenguaje principal.

El ejemplo siguiente muestra una aplicación con dos instancias abiertas simultáneamente de un iterador.

```
...
ResultSet myFunc(String empid)  // Método para abrir un iterador y
                                // obtener un conjunto de resultados
{
  MyIter iter;
  #sql iter = {SELECT * FROM EMPLOYEE WHERE EMPNO = :empid};
  return iter.getResultSet();
}

// Una aplicación puede invocar este método para obtener un conjunto
// de resultados para cada ID de empleado. La aplicación puede
// procesar cada conjunto de resultados por separado.
...
ResultSet rs1 = myFunc("000100");  // Obtener registro del ID de empleado 000100
...
ResultSet rs2 = myFunc("000200");  // Obtener registro del ID de empleado 000200
```

Figura 47. Ejemplo de apertura de más de una instancia de un iterador en una misma aplicación

Al igual que ocurre con cualquier otro iterador, es necesario cerrar este iterador cuando termine de utilizarlo para evitar un uso excesivo de espacio de almacenamiento.

Conceptos relacionados:

- "Recuperación de datos de tablas DB2 por una aplicación SQLJ" en la página 357

Invocación de procedimientos almacenados en una aplicación SQLJ

Para invocar un procedimiento almacenado, utilice una cláusula ejecutable que contenga una sentencia CALL de SQL. Puede ejecutar la sentencia CALL con parámetros de identificador de lenguaje principal. Estos son los pasos básicos para invocar un procedimiento almacenado:

1. Asigne valores a los parámetros de entrada (IN o INOUT).
2. Invoque el procedimiento almacenado.
3. Procese los parámetros de salida (OUT o INOUT).
4. Si el procedimiento almacenado devuelve varios conjuntos de resultados, obtenga esos resultados. Consulte el tema Recuperar varios conjuntos de resultados de un procedimiento almacenado en una aplicación SQLJ.

El código siguiente muestra la invocación de un procedimiento almacenado que tiene tres parámetros de entrada y tres parámetros de salida. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
String FirstName="TOM";       // Parámetros de entrada    1
String LastName="NARISINST";
String Address="IBM";
int CustNo;                          // Parámetros de salida
String Mark;
String MarkErrorText;
...
#sql [myConnCtx] {CALL ADD_CUSTOMER(:IN FirstName,                    2
                            :IN LastName,
                            :IN Address,
                            :OUT CustNo,
                            :OUT Mark,
                            :OUT MarkErrorText)};
                          // Invocar el procedimiento almacenado
System.out.println("Output parameters from ADD_CUSTOMER call: ");
System.out.println("Customer number for " + LastName + ": " + CustNo);  3
System.out.println(Mark);
If (MarkErrorText != null)
  System.out.println(" Error messages:" + MarkErrorText);
```

Figura 48. Invocación de un procedimiento almacenado en una aplicación SQLJ

Conceptos relacionados:

- "Recuperación de varios conjuntos de resultados de un procedimiento almacenado en una aplicación SQLJ" en la página 382

Manejo de errores de SQL en una aplicación SQLJ

Las cláusulas de SQLJ utilizan la clase `java.sql.SQLException` de JDBC para el manejo de errores. SQLJ genera una excepción `SQLException` en las condiciones siguientes:

- Cuando una sentencia cualquiera de SQL devuelve un código de error SQL negativo
- Cuando una sentencia SELECT INTO de SQL devuelve el código de error +100 de SQL

Puede utilizar el método `getErrorCode` para obtener los códigos de error SQL, y el método `getSQLState` para obtener los SQLSTATE (estados de SQL).

Para el manejo de errores SQL en su aplicación SQLJ, importe la clase `java.sql.SQLException` y utilice los bloques `try/catch` de manejo de errores de Java™ para modificar el flujo del programa cuando se produzca un error de SQL. Por ejemplo:

```
try {
  #sql [ctxt] {SELECT LASTNAME INTO :empname
    FROM EMPLOYEE WHERE EMPNO='000010'};
}
 catch (SQLException e) {
  System.out.println("Código de error devuelto: " + e.getErrorCode());
}
```

Cuando se utiliza el Controlador JDBC universal de DB2, puede recuperar el contenido de la SQLCA. Para conocer cómo escribir un código de programación que obtenga el contenido de la SQLCA cuando se utiliza el Controlador JDBC universal de DB2, consulte el tema Manejo de una excepción de SQL cuando se utiliza el Controlador JDBC de DB2 Universal.

Para el Controlador JDBC de DB2 de tipo 2 para Linux, UNIX® y Windows® (Controlador JDBC de DB2 de tipo 2), utilice el objeto estándar `SQLException` para obtener información sobre errores de SQL.

Tareas relacionadas:

- "Manejo de una excepción de SQL cuando se utiliza el Controlador JDBC de DB2 Universal" en la página 308

Manejo de avisos de SQL en una aplicación SQLJ

Excepto el código de error +100 de SQL para una sentencia SELECT INTO, los avisos de DB2® no emiten excepciones de SQL. Para manejar los avisos de DB2, debe otorgar al programa acceso a la clase `java.sql.SQLWarning`. Si desea recuperar información sobre un aviso que sea específica de DB2, es necesario también que otorgue al programa acceso a la interfaz `com.ibm.db2.jcc.DB2Diagnosable` y a la clase `com.ibm.db2.jcc.DB2Sqlca`. Para comprobar la existencia de avisos de DB2, invoque el método `getWarnings` después de ejecutar una cláusula de SQLJ. `getWarnings` devuelve el primer objeto `SQLWarning` generado por una sentencia de SQL. Los objetos `SQLWarning` subsiguientes se encadenan al primero de ellos.

Para recuperar información específica de DB2 a partir del objeto `SQLWarning` cuando se utiliza el Controlador JDBC universal de DB2, siga las instrucciones del tema Manejo de una excepción de SQL cuando se utiliza el Controlador JDBC de DB2 Universal.

Para ejecutar `getWarnings` para una cláusula de SQL, es necesario que defina un contexto de ejecución para esa cláusula de SQL. Consulte el tema Control de la ejecución de sentencias de SQL en SQLJ para obtener información sobre cómo definir un contexto de ejecución. El ejemplo siguiente muestra cómo recuperar un objeto `SQLWarning` para una cláusula de SQL con el contexto de ejecución execCtx:

```
ExecutionContext execCtx=myConnCtx.getExecutionContext();
                             // Obtener el contexto de ejecución
                             // por omisión a partir del contexto
                             // de conexión
SQLWarning sqlWarn;
...
#sql [myConnCtx,execCtx] {SELECT LASTNAME INTO :empname
  FROM EMPLOYEE WHERE EMPNO='000010'};
if ((sqlWarn = execCtx.getWarnings()) != null)
System.out.println("SQLWarning " + sqlWarn);
```

Tareas relacionadas:

- "Manejo de una excepción de SQL cuando se utiliza el Controlador JDBC de DB2 Universal" en la página 308
- "Control de la ejecución de sentencias de SQL en SQLJ" en la página 381
- "Manejo de errores de SQL en una aplicación SQLJ" en la página 371

Conceptos avanzados de la programación de aplicaciones SQLJ

Los temas siguientes contienen información más compleja sobre la escritura de aplicaciones SQLJ.

Uso de SQLJ y JDBC en la misma aplicación

Puede combinar cláusulas SQLJ y llamadas JDBC en un mismo programa. Para ello, debe seguir estos pasos:

- Utilice una `conexión` JDBC para crear `contexto de conexión` SQLJ u obtenga una `conexión` JDBC a partir de un `contexto de conexión` SQLJ.
- Utilice un iterador SQLJ para obtener datos a partir de un `conjunto de resultados` JDBC o genere un `conjunto de resultados` JDBC a partir de un iterador SQLJ.

Creación de un contexto de conexión SQLJ a partir de una conexión JDBC - Siga estos pasos:

1. Ejecute una cláusula de declaración de conexión SQLJ para crear una clase `ConnectionContext`.
2. Cargue el controlador u obtenga una instancia de `DataSource`.
3. Invoque el método `DriverManager.getConnection` o `DataSource.getConnection` de JDBC para obtener una `conexión` JDBC.
4. Invoque el constructor `ConnectionContext` con el objeto `Connection` como argumento para crear el objeto `ConnectionContext`.

Obtención de una conexión JDBC a partir de una contexto de conexión SQLJ - Siga estos pasos:

1. Ejecute una cláusula de declaración de conexión SQLJ para crear una clase `ConnectionContext`.
2. Cargue el controlador u obtenga una instancia de `DataSource`.
3. Invoque el constructor `ConnectionContext` con el URL del controlador y cualquier otro parámetro necesario como argumentos para crear el objeto `ConnectionContext`.
4. Invoque el método `ConnectionContext.getConnection` de JDBC para crear el objeto `Connection` de JDBC.

Consulte el tema Conectar a una fuente de datos utilizando SQLJ para obtener más información sobre las conexiones SQLJ.

Obtención de conjuntos de resultados JDBC utilizando iteradores SQLJ: Utilice la *sentencia de conversión a iterador* para manejar un conjunto de resultados JDBC como un iterador SQLJ. Este es el formato general de una sentencia de conversión a iterador:

```
#sql iterador={CAST :conjunto-resultados};
```

Para convertir satisfactoriamente un conjunto de resultados en un iterador, éste debe cumplir las reglas siguientes:

- El iterador debe estar declarado como público.
- Si el iterador es un iterador de posición, el número de columnas del conjunto de resultados debe coincidir con el número de columnas del iterador. Además, el tipo de datos de cada columna del conjunto de resultados debe coincidir con el tipo de datos de la columna correspondiente del iterador.

- Si el iterador es un iterador de nombre, el nombre de cada método accesor debe coincidir con el nombre de una columna del conjunto de resultados. Además, el tipo de datos del objeto devuelto por un método accesor debe coincidir con el tipo de datos de la columna correspondiente del conjunto de resultados.

El código mostrado en la Figura 49 crea y ejecuta una consulta utilizando una llamada JDBC, ejecuta una sentencia de conversión a iterador para convertir el conjunto de resultados JDBC en un iterador SQLJ y obtiene filas del conjunto de resultados utilizando el iterador.

```
#sql public iterator ByName(String LastName, Date HireDate);      1
public void HireDates(ConnectionContext connCtx, String whereClause)
{

  ByName nameiter;       // Declarar objeto de la clase ByName
  Connection conn=connCtx.getConnection();
                         // Crear la conexión JDBC
  Statement stmt = conn.createStatement();                         2
  String query = "SELECT LASTNAME, HIREDATE FROM EMPLOYEE";
  query+=whereClause;   // Crear la consulta
  ResultSet rs = stmt.executeQuery(query);                         3
  #sql [connCtx] nameiter = {CAST :rs};                            4
  while (nameiter.next())
  {
    System.out.println( nameiter.LastName() + " was hired on "
      + nameiter.HireDate());

  }
  nameiter.close();                                                5
stmt.close();
}
```

Figura 49. Conversión de un conjunto de resultados JDBC en un iterador SQLJ

Nota para la Figura 49:

1 Este cláusula de SQLJ crea la clase de iterador de nombre ByName, cuyos métodos accesores LastName() y HireDate() obtienen datos de las columnas LASTNAME y HIREDATE de la tabla de resultados.

2 Esta sentencia y las dos sentencias que le siguen crean y preparan una consulta para su ejecución dinámica mediante JDBC.

3 Esta sentencia de JDBC ejecuta la sentencia SELECT y asigna la tabla de resultados al conjunto de resultados rs.

4 Esta cláusula de conversión a iterador convierte el `conjunto de resultados` rs de JDBC en el iterador nameiter de SQLJ y las sentencias que siguen utilizan nameiter para obtener valores de la tabla de resultados.

5 El método `nameiter.close()` cierra el iterador de SQLJ y el `conjunto de resultados` rs de JDBC.

Generación de conjuntos de resultados de JDBC a partir de iteradores de SQLJ: utilice el método getResultSet para generar un `conjunto de resultados` a partir de un iterador de SQLJ. Cada iterador de SQLJ tiene un método getResultSet. Después de convertir un iterador en un conjunto de resultados, necesita recuperar filas utilizando solamente el conjunto de resultados.

El código mostrado en la Figura 50 en la página 375 genera un iterador de posición para una consulta, convierte el iterador en un conjunto de resultados y utiliza métodos de JDBC para recuperar filas de la tabla.

```
#sql iterator EmpIter(String, java.sql.Date);
{
...
  EmpIter iter=null;
  #sql [connCtx] iter=
    {SELECT LASTNAME, HIREDATE FROM EMPLOYEE};                1
  ResultSet rs=iter.getResultSet();                           2
  while (rs.next())                                           3
  { System.out.println(rs.getString(1) + " was hired in " +
      rs.getDate(2));
  }
  rs.close();                                                 4
}
```

Figura 50. Conversión de un iterador SQLJ en un conjunto de resultados JDBC

Nota para la Figura 50:

1 Esta cláusula de SQLJ ejecuta la sentencia SELECT, construye un objeto iterador que contiene la tabla de resultados correspondiente a la sentencia SELECT y asigna el objeto iterador a la variable iter.

2 El método getResultSet() convierte el iterador iter en el `conjunto de resultados` rs.

3 Los métodos getString() y getDate() de JDBC obtienen valores a partir del `conjunto de resultados`. El método next() coloca el cursor en la fila siguiente del `conjunto de resultados`.

4 El método `rs.close()` cierra el iterador de SQLJ y el `conjunto de resultados`.

Reglas y restricciones para utilizar conjuntos de resultados de JDBC en aplicaciones SQLJ: tenga en cuenta las reglas y restricciones siguientes al escribir aplicaciones de SQLJ que incluyen conjuntos de resultados de JDBC:

- No puede convertir un `conjunto de resultados` en un iterador de SQLJ si el `conjunto de resultados` y el iterador tienen valores diferentes para el atributo de capacidad de retención del cursor.

  Un `conjunto de resultados` de JDBC o iterador de SQLJ pueden permanecer abiertos después de una operación COMMIT. Para un `conjunto de resultados` de JDBC, esta característica está controlada por la propiedad `resultSetHoldability` del Controlador JDBC universal de DB2. Para un iterador de SQLJ, esta característica está controlada por el parámetro `with holdability` de la sentencia de declaración del iterador. No está soportada la conversión de un `conjunto de resultados` con capacidad de retención en un iterador de SQLJ que no tenga esa capacidad, ni la conversión de un `conjunto de resultados` sin capacidad de retención en un iterador que sí la tenga.

- Cierre el objeto `ResultSet` generado o el iterador subyacente al final del programa.

  Cuando cierra el objeto iterador utilizado para crear un objeto `ResultSet`, también se cierra el objeto `ResultSet`. Cuando cierra el objeto `ResultSet` generado, también se cierra el objeto iterador. En general, es mejor cerrar el objeto que se ha utilizado en último lugar.

- Para el Controlador JDBC universal de DB2, que da soporte a iteradores desplazables y a `conjuntos de resultados` desplazables y actualizables, son aplicables las restricciones siguientes:
  - Los iteradores desplazables tienen las mismas restricciones que sus `conjuntos de resultados` de JDBC subyacentes. Por ejemplo, debido a que los `conjuntos de resultados` desplazables no dan soporte a las operaciones INSERT, los iteradores desplazables no dan soporte a las operaciones INSERT.

– No puede convertir un `conjunto de resultados` de JDBC que no sea actualizable en un iterador SQLJ que sea actualizable.

Tareas relacionadas:

- "Conexión a una fuente de datos utilizando SQLJ" en la página 349

Datos LOB en aplicaciones JDBC con el Controlador JDBC de DB2 Universal

Con el Controlador JDBC universal de DB2, puede insertar datos LOB en expresiones `Clob` o `Blob` de lenguaje principal o actualizar columnas CLOB, BLOB o DBCLOB a partir de expresiones `Clob` o `Blob` de lenguaje principal. Puede también declarar iteradores con tipos de datos `Clob` o `Blob` para recuperar datos de columnas CLOB, BLOB o DBCLOB.

Recuperación o actualización de datos LOB: para recuperar datos de una columna BLOB, declare un iterador que incluya el tipo de datos `Blob` o `byte[]`. Para recuperar datos de una columna CLOB o DBCLOB, declare un iterador en el que la correspondiente columna tenga un tipo de datos `Clob`.

Para actualizar datos de una columna BLOB, utilice una expresión de lenguaje principal cuyo tipo de datos sea `Blob`. Para actualizar datos de una columna CLOB o DBCLOB, utilice una expresión de lenguaje principal cuyo tipo de datos sea `Clob`.

Soporte de localizador de LOB: el Controlador JDBC universal de DB2 puede utilizar localizadores de LOB para recuperar datos. Para que JDBC utilice localizadores de LOB para recuperar datos de columnas LOB, debe establecer la propiedad `fullyMaterializeLobData` en `false`. Las propiedades se describen en el tema Propiedades del Controlador JDBC de DB2® Universal.`fullyMaterializeLobData` no tiene ningún efecto para parámetros de salida de procedimiento almacenado ni datos LOB que se recuperan utilizando cursores desplazables. No puede invocar un procedimiento almacenado que tenga parámetros de localizador de LOB. Cuando realiza recuperaciones de datos utilizando cursores desplazables, JDBC siempre utiliza localizadores de LOB para recuperar datos de columnas LOB.

Al igual que en cualquier otro lenguaje, en una aplicación Java un localizador de LOB está asociado a una sola base de datos.. No puede utilizar un localizador de LOB individual para trasladar datos entre dos bases de datos diferentes. Para trasladar datos entre dos bases de datos, es necesario materializar los datos LOB cuando los recupera de una tabla de la primera base de datos y luego insertar esos datos en la tabla de la segunda base de datos.

Información relacionada:

- "Propiedades del Controlador JDBC de DB2 Universal" en la página 400
- "Tipos de datos de Java, JDBC y SQL" en la página 395

Tipos de datos Java para recuperar o actualizar datos de columnas LOB en aplicaciones SQLJ

Cuando la propiedad deferPrepares tiene el valor "true", y el Controlador JDBC universal de DB2 procesa una sentencia de SQLJ no personalizada que incluye expresiones de lenguaje principal, el controlador puede necesitar realizar tareas

adicionales de proceso para determinar los tipos de datos. Estas tareas adicionales de proceso pueden afectar al rendimiento del sistema.

Cuando el controlador JDBC no puede determinar inmediatamente el tipo de datos de un parámetro que se utiliza con una columna LOB, es necesario elegir un tipo de datos para el parámetro que sea compatible con el tipo de datos LOB.

Cuando el controlador JDBC no puede determinar el tipo de datos de un parámetro que se utiliza con una columna LOB, es necesario elegir un tipo de datos para el parámetro que sea compatible con el tipo de datos LOB.

Parámetros de entrada para columnas BLOB:

Para los parámetros de entrada de columnas BLOB, puede utilizar cualquiera de las dos técnicas siguientes:

- Utilice una variable de entrada java.sql.Blob, que se corresponde exactamente con una columna BLOB:

  ```
  java.sql.Blob blobData;
  #sql {CALL STORPROC(:IN blobData)};
  ```

  Para poder utilizar una variable de entrada java.sql.Blob, es necesario crear un objeto java.sql.Blob y luego llenar con datos ese objeto. Por ejemplo, si está utilizando el Controlador JDBC universal de DB2, puede utilizar el método `com.ibm.db2.jcc.t2zos.DB2LobFactory.createBlob`, exclusivo de DB2, para llenar el objeto con datos de tipo byte[]:

  ```
  byte[] byteArray = {0, 1, 2, 3};
  java.sql.Blob blobData =
    com.ibm.db2.jcc.t2zos.DB2LobFactory.createBlob(byteArray);
  ```

- Utilice un parámetro de entrada de tipo `sqlj.runtime.BinaryStream`. Un objeto `sqlj.runtime.BinaryStream` es compatible con un tipo de datos BLOB. Para esta llamada es necesario especificar la longitud exacta de los datos de entrada:

  ```
  java.io.ByteArrayInputStream byteStream =
    new java.io.ByteArrayInputStream(byteData);
  int numBytes = byteData.length;
  sqlj.runtime.BinaryStream binStream =
    new sqlj.runtime.BinaryStream(byteStream, numBytes);
  #sql {CALL STORPROC(:IN binStream)};
  ```

  No puede utilizar esta técnica para parámetros de entrada/salida.

Parámetros de salida para columnas BLOB:

Para los parámetros de salida o entrada/salida de columnas BLOB, puede utilizar la técnica siguiente:

- Declare el parámetro de salida o variable de entrada/salida con un tipo de datos java.sql.Blob:

  ```
  java.sql.Blob blobData = null;
  #sql CALL STORPROC (:OUT blobData)};
  ```

  ```
  java.sql.Blob blobData = null;
  #sql CALL STORPROC (:INOUT blobData)};
  ```

Parámetros de entrada para columnas CLOB:

Para los parámetros de entrada de columnas CLOB, puede utilizar una de las técnicas siguientes:

- Utilice una variable de entrada java.sql.Clob, que se corresponde exactamente con una columna CLOB:

```
#sql CALL STORPROC(:IN clobData)};
```

  Para poder utilizar una variable de entrada java.sql.Clob, es necesario crear un objeto java.sql.Clob y luego llenar con datos ese objeto. Por ejemplo, si está utilizando el Controlador JDBC universal de DB2, puede utilizar el método `com.ibm.db2.jcc.t2zos.DB2LobFactory.createClob`, exclusivo de DB2, para llenar el objeto con datos de tipo String:

```
String
stringVal = "Algunos datos";
java.sql.Clob clobData =
  com.ibm.db2.jcc.t2zos.DB2LobFactory.createClob(stringVal);
```

- Utilice uno de los tipos siguientes de parámetros de entrada:
  - Un parámetro de entrada `sqlj.runtime.CharacterStream`:

```
java.lang.String charData;
java.io.StringReader reader = new java.io.StringReader(charData);
sqlj.runtime.CharacterStream charStream =
  new sqlj.runtime.CharacterStream (reader, charData.length);
#sql {CALL STORPROC(:IN charStream)};
```

  - Un parámetro `sqlj.runtime.UnicodeStream`, para datos Unicode UTF-16:

```
byte[] charDataBytes = charData.getBytes("UnicodeBigUnmarked");
java.io.ByteArrayInputStream byteStream =
  new java.io.ByteArrayInputStream(charDataBytes);
sqlj.runtime.UnicodeStream uniStream =
  new sqlj.runtime.UnicodeStream(byteStream, charDataBytes.length );
#sql {CALL STORPROC(:IN uniStream)};
```

  - Un parámetro `sqlj.runtime.AsciiStream`, para datos ASCII:

```
byte[] charDataBytes = charData.getBytes("US-ASCII");
java.io.ByteArrayInputStream byteStream =
  new java.io.ByteArrayInputStream (charDataBytes);
sqlj.runtime.AsciiStream asciiStream =
  new sqlj.runtime.AsciiStream (byteStream, charDataBytes.length);
#sql {CALL STORPROC(:IN asciiStream)};
```

  Para estas llamadas es necesario especificar la longitud exacta de los datos de entrada. No puede utilizar esta técnica para parámetros de entrada/salida.
- Utilice un parámetro de entrada java.lang.String:

```
java.lang.String charData;
#sql {CALL STORPROC(:IN charData)};
```

Parámetros de salida para columnas CLOB:

Para los parámetros de salida o entrada/salida de columnas CLOB, puede utilizar una de las técnicas siguientes:

- Utilice una variable de salida java.sql.Clob, que se corresponde exactamente con una columna CLOB:

```
java.sql.Clob clobData = null;
#sql CALL STORPROC(:OUT clobData)};
```

- Utilice una variable de salida java.lang.String:

```
java.lang.String charData = null;
#sql CALL STORPROC(:OUT charData)};
```

  Esta técnica solo se debe utilizar si conoce que la longitud de los datos recuperados es menor o igual que 32 KB. En otro caso, los datos se truncan.

Parámetros de salida para columnas DBCLOB:

Los parámetros de salida o entrada/salida de DBCLOB para procedimientos almacenados no están soportados.

Conceptos relacionados:

- "Datos LOB en aplicaciones JDBC con el Controlador JDBC de DB2 Universal" en la página 376

Información relacionada:

- "Tipos de datos de Java, JDBC y SQL" en la página 395

Uso de valores ROWID en JDBC con el Controlador JDBC universal de DB2

DB2® UDB para z/OS® y DB2 UDB para iSeries™ dan soporte al tipo de datos ROWID para una columna de una tabla DB2. Un ROWID es un valor que identifica una fila de una tabla de una forma exclusiva.

Si utiliza valores ROWID en programas SQLJ, es necesario que personalice esos programas.

El Controlador JDBC universal de DB2 proporciona la clase `com.ibm.db2.jcc.DB2RowID`, exclusiva de DB2, que puede utilizar en iteradores y en parámetros de sentencias CALL. En el caso de un iterador, puede también utilizar el tipo de objeto byte[] para recuperar valores ROWID.

La Figura 51 en la página 380 muestra un ejemplo de un iterador que se utiliza para seleccionar valores de una columna ROWID:

```
#sql iterator PosIter(int,String,com.ibm.db2.jcc.DB2RowId);
                    // Declarar un iterador de posición para
                    // recuperar valores ITEM_ID (INTEGER),
                    // ITEM_FORMAT (VARCHAR) y ITEM_ROWID (ROWID)
                    // de la tabla ROWIDTAB
{
  PosIter positrowid;       // Declarar objeto de la clase PosIter
  com.ibm.db2.jcc.DB2RowId rowid = null;
  int id = 0;
  String i_fmt = null;
                            // Declarar expresiones de lenguaje principal
  #sql [ctxt] positrowid =
    {SELECT ITEM_ID, ITEM_FORMAT, ITEM_ROWID FROM ROWIDTAB
       WHERE ITEM_ID=3};
                            // Asignar tabla de resultados de SELECT
                            // al objeto de iterador positrowid
  #sql {FETCH :positrowid INTO :id, :i_fmt, :rowid};
                            // Recuperar primera fila
  while (!positrowid.endFetch())
                           // Determinar si FETCH devolvió una fila
    {System.out.println("Item ID " + id + " Item format " +
       i_fmt + " Item ROWID ");
     printBytes(rowid.getBytes());
                         // Utilizar el método getBytes, exclusivo de DB2,
                         // para convertir el valor a bytes para imprimir
      #sql {FETCH :positrowid INTO :id, :i_fmt, :rowid};
                              // Recuperar la fila siguiente
    }
  positrowid.close();          // Cerrar el iterador
}
```

Figura 51. Ejemplo de utilización de un iterador para recuperar valores ROWID

La Figura 52 muestra un ejemplo de invocación de un procedimiento almacenado que utiliza tres parámetros ROWID: un parámetro de entrada (IN), un parámetro de salida (OUT) y un parámetro de entrada/salida (INOUT).

```
com.ibm.db2.jcc.DB2RowId in_rowid = rowid;
com.ibm.db2.jcc.DB2RowId out_rowid = null;
com.ibm.db2.jcc.DB2RowId inout_rowid = rowid;
                         // Declarar un parámetro ROWID de
                         // entrada, salida, entrada/salida
...
#sql [myConnCtx] {CALL SP_ROWID(:IN in_rowid,
                            :OUT out_rowid,
                            :INOUT inout_rowid)};
                         // Invocar el procedimiento almacenado
System.out.println("Parameter values from SP_ROWID call: ");
System.out.println("Output parameter value ");
printBytes(out_rowid.getBytes());
                         // Utilizar el método getBytes, exclusivo de DB2,
                         // para convertir el valor a bytes para imprimir
System.out.println("Input/output parameter value ");
printBytes(inout_rowid.getBytes());
```

Figura 52. Ejemplo de invocación de un procedimiento almacenado con un parámetro ROWID

Información relacionada:

- "Tipos de datos de Java, JDBC y SQL" en la página 395

Tipos diferenciados en aplicaciones SQLJ

En DB2®, un tipo diferenciado es un tipo de datos definido por el usuario cuya representación interna es un tipo de datos SQL incorporado. Un tipo diferenciado se crea ejecutando la sentencia CREATE DISTINCT TYPE de SQL.

En un programa SQLJ, puede crear un tipo diferenciado utilizando la sentencia CREATE DISTINCT TYPE en una cláusula ejecutable. Puede también utilizar CREATE TABLE en una cláusula ejecutable para crear una tabla que incluya una columna de ese tipo. Cuando recupera datos de una columna de ese tipo, o actualiza una columna de ese tipo, utiliza identificadores de Java™ con tipos de datos que corresponden a los tipos incorporados en los que se basan los tipos diferenciados.

El ejemplo siguiente crea un tipo diferenciado que está basado en un tipo INTEGER, crea una tabla con una columna de ese tipo, inserta una fila en la tabla y recupera la fila de la tabla:

```
String empNumVar;
int shoeSizeVar;
...
#sql [myConnCtx] {CREATE DISTINCT TYPE SHOESIZE AS INTEGER WITH COMPARISONS};
                                  // Crear tipo diferenciado
#sql [myConnCtx] {COMMIT}; // Confirmar la creación
#sql [myConnCtx] {CREATE TABLE EMP_SHOE
  (EMPNO CHAR(6), EMP_SHOE_SIZE SHOESIZE)};
                                 // Crear tabla utilizando tipo diferenciado
#sql [myConnCtx] {COMMIT}; // Confirmar la creación
#sql [myConnCtx] {INSERT INTO EMP_SHOE
  VALUES('000010',6)};         // Insertar una fila en la tabla
#sql [myConnCtx] {COMMIT}; // Confirmar la inserción
#sql [myConnCtx] {SELECT EMPNO, EMP_SHOE_SIZE
  INTO :empNumVar, :shoeSizeVar
  FROM EMP_SHOE};                     // Recuperar la fila
System.out.println("Employee number: " + empNumVar +
  " Shoe size: " + shoeSizeVar);
```

Figura 53. Definición y utilización de un tipo diferenciado

Información relacionada:

- "Sentencia CREATE DISTINCT TYPE" en la publicación *Consulta de SQL, Volumen 2*

Control de la ejecución de sentencias de SQL en SQLJ

Puede utilizar determinados métodos de la clase `ExecutionContext` de SQLJ para controlar o supervisar la ejecución de sentencias de SQL. El tema Clases e interfaces seleccionadas de sqlj.runtime describe esos métodos.

Siga estos pasos para utilizar métodos de `ExecutionContext`:

1. Adquiera un *contexto de ejecución*.

   Existen dos formas de adquirir un contexto de ejecución:

   - Adquiera el contexto de ejecución por omisión a partir del contexto de conexión. Por ejemplo:

     ```
     ExecutionContext execCtx = connCtx.getExecutionContext();
     ```

- Cree un nuevo contexto de ejecución invocando el constructor de `ExecutionContext`. Por ejemplo:

  ```
  ExecutionContext execCtx=new ExecutionContext();
  ```

2. Asocie el contexto de ejecución a una sentencia de SQL.

   Para ello, especifique un contexto de ejecución a continuación del contexto de conexión en la cláusula de ejecución donde reside la sentencia de SQL. Por ejemplo:

   ```
   #sql [connCtx, execCtx] {DELETE FROM EMPLOYEE WHERE SALARY > 10000};
   ```

3. Invoque métodos de `ExecutionContext`.

   Algunos métodos de ExecutionContext son aplicables antes de ejecutar la sentencia de SQL asociada, mientras que otros son aplicables solo después de ejecutar su sentencia de SQL asociada.

   Por ejemplo, puede utilizar el método `getUpdateCount` para contar el número de filas suprimidas por una sentencia DELETE después de ejecutar esa sentencia:

   ```
   #sql [connCtx, execCtx] {DELETE FROM EMPLOYEE WHERE SALARY > 10000};
   System.out.println("Deleted " + execCtx.getUpdateCount() + " rows");
   ```

Información relacionada:

• "Clases e interfaces selectas de sqlj.runtime" en la página 438

Recuperación de varios conjuntos de resultados de un procedimiento almacenado en una aplicación SQLJ

Algunos procedimientos almacenados devuelven uno o más conjuntos de resultados al programa solicitante. Para recuperar las filas de esos conjuntos de resultados, siga estos pasos:

1. Adquiera un contexto de ejecución para recuperar el conjunto de resultados del procedimiento almacenado.
2. Asocie el contexto de ejecución con la sentencia CALL para el procedimiento almacenado.

   No utilice este contexto de ejecución para ninguna otra finalidad hasta que haya recuperado y procesado el último conjunto de resultados.
3. Para cada conjunto de resultados:
   a. Utilice el método `getNextResultSet` del `contexto de ejecución` para recuperar el conjunto de resultados.
   b. Si no conoce el contenido del conjunto de resultados, utilice el método `ResultSetMetaData` para obtener esta información.
   c. Utilice un iterador de conjunto de resultados de SQLJ o un `conjunto de resultados` de JDBC para recuperar las filas del conjunto de resultados.

Los conjuntos de resultados se devuelven al programa solicitante en el mismo orden en el que se abren los cursores del programa en el procedimiento almacenado. Cuando no existen más conjuntos de resultados para recuperar, getNextResultSet devuelve un valor nulo.

Existen dos formatos de `getNextResultSet`:

```
getNextResultSet();
getNextResultSet(int actual);
```

Cuando invoca el primer formato de `getNextResultSet`, SQLJ cierra el conjunto de resultados que está abierto actualmente y pasa al conjunto de resultados siguiente.

Cuando invoca el segundo formato de `getNextResultSet`, el valor de *actual* indica lo que SQLJ realiza con el conjunto de resultados abierto actualmente antes de pasar al conjunto de resultados siguiente:

java.sql.Statement.CLOSE_CURRENT_RESULT
Especifica que el objeto `ResultSet` actual se cierra cuando se devuelve el objeto `ResultSet` siguiente.

java.sql.Statement.KEEP_CURRENT_RESULT
Especifica que el objeto `ResultSet` actual permanece abierto cuando se devuelve el objeto `ResultSet` siguiente.

java.sql.Statement.CLOSE_ALL_RESULTS
Especifica que todos los objetos `ResultSet` abiertos se cierran cuando se devuelve el objeto `ResultSet` siguiente.

El segundo formato de `getNextResultSet` requiere utilizar JDK 1.4 o una versión posterior.

El código siguiente invoca un procedimiento almacenado que devuelve varios conjuntos de resultados. En este ejemplo se supone que el usuario solicitante no conoce el número de conjuntos de resultados a devolver ni el contenido de esos conjuntos de resultados. Se supone también que autoCommit tiene el valor ″false″. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
ExecutionContext execCtx=myConnCtx.getExecutionContext();       1
#sql [myConnCtx, execCtx] {CALL MULTRSSP()};                    2
            // MULTRSSP devuelve varios conjuntos de resultados
    ResultSet rs;
while ((rs = execCtx.getNextResultSet()) != null)               3a
{
  ResultSetMetaData rsmeta=rs.getMetaData();                    3b
  int numcols=rsmeta.getColumnCount();
  while (rs.next())                                             3c
  {
    for (int i=1; i<=numcols; i++)
    {
      String colval=rs.getString(i);
      System.out.println("Column " + i + "value is " + colval);
    }
  }
}
```

Figura 54. Recuperación de conjuntos de resultados de un procedimiento almacenado

Realización de actualizaciones por lotes en aplicaciones SQLJ

El Controlador JDBC universal de DB2 soporta la realización de actualizaciones por lotes en SQLJ. Mediante las actualizaciones por lotes, en lugar de actualizar una sola fila cada vez en una tabla DB2®, puede hacer que SQLJ ejecute varias actualizaciones al mismo tiempo. Puede incluir los tipos siguientes de sentencias en una actualización por lotes:

- Sentencias INSERT, UPDATE y DELETE de búsqueda
- Sentencias CREATE, ALTER, DROP, GRANT y REVOKE
- Sentencias CALL con parámetros de entrada solamente

A diferencia de JDBC, SQLJ permite utilizar lotes heterogéneos que contienen sentencias con parámetros de entrada o expresiones de lenguaje principal. Puede,

por tanto, combinar instancias de la misma sentencia, sentencias diferentes, sentencias con parámetros de entrada o expresiones de lenguaje principal, y sentencias sin parámetros de entrada ni expresiones de lenguaje principal en un mismo lote de sentencias de SQLJ.

Estos son los pasos básicos para crear, ejecutar y suprimir un lote de sentencias:

1. Inhabilite `AutoCommit` (confirmación automática) para la conexión.
2. Obtenga un contexto de ejecución.

   Todas las sentencias que se ejecutan en un lote deben utilizar este contexto de ejecución.
3. Invoque el método `ExecutionContext.setBatching(true)` para crear un lote.

   Las subsiguientes sentencias procesables por lotes que están asociadas al contexto de ejecución creado en el paso 2 se añaden al lote para su ejecución posterior.

   Si desea procesar por lotes conjuntos de sentencias que no son compatibles respecto al proceso por lotes en paralelo, debe crear un contexto de ejecución para cada conjunto de sentencias compatibles respecto al proceso por lotes.
4. Incluya cláusulas ejecutables de SQLJ para las sentencias de SQL que desee procesar por lotes.

   Estas cláusulas deben incluir el contexto de ejecución que creó en el paso 2.

   Si una cláusula ejecutable de SQLJ tiene parámetros de entrada o expresiones de lenguaje principal, puede incluir la sentencia en el lote varias veces con valores diferentes para los parámetros de entrada o expresiones de lenguaje principal.

   Para determinar si una sentencia se añadió a un lote existente, si era la primera sentencia de un nuevo lote, o si se ejecutó dentro o fuera de un lote, invoque el método `ExecutionContext.getUpdateCount`. Este método devuelve uno de los valores siguientes:

   `ExecutionContext.ADD_BATCH_COUNT`
   Se devuelve esta constante si la sentencia se añadió a un lote existente.

   `ExecutionContext.NEW_BATCH_COUNT`
   Se devuelve esta constante si la sentencia era la primera sentencia de un nuevo lote.

   `ExecutionContext.EXEC_BATCH_COUNT`
   Se devuelve esta constante si la sentencia formaba parte de un lote que se ejecutó.

   Otro valor entero
   Este valor es el número de filas que fueron actualizadas por la sentencia. Se devuelve este valor si se ejecutó la sentencia en lugar de añadirla a un lote.
5. Ejecute el lote explícita o implícitamente.
   - Invoque el método `ExecutionContext.executeBatch` para ejecutar el lote explícitamente.

     `executeBatch` devuelve una matriz entera que contiene el número de filas que fueron actualizadas por cada sentencia del lote. El orden de los elementos de la matriz corresponde al orden en el que se añadieron las sentencias al lote.
   - Como alternativa, un lote se ejecuta implícitamente en las condiciones siguientes:
     – Cuando incluye en su programa una sentencia procesable por lotes que no es compatible con sentencias ya existentes en el lote. En este caso, SQLJ

ejecuta las sentencias que ya existen en el lote y crea un nuevo lote donde se incluye la sentencia incompatible. SQLJ también ejecuta la sentencia que no es compatible con las sentencias del lote.

– Cuando incluye en su programa una sentencia que no es ejecutable por lotes. En este caso, SQLJ ejecuta las sentencias que ya existen en el lote. SQLJ también ejecuta la sentencia que no es procesable por lotes.
– Cuando después de invocar el método `ExecutionContext.setBatchLimit(n)`, añade una sentencia al lote que hace que el número de sentencias del lote sea igual o mayor que *n*.*n* puede tener uno de los valores siguientes:

  `ExecutionContext.UNLIMITED_BATCH`
  Esta constante indica que la ejecución implícita solo se produce cuando SQLJ encuentra una sentencia que es procesable por lotes pero incompatible, o que no es procesable por lotes. Establecer este valor es lo mismo que no invocar `setBatchLimit`.

  `ExecutionContext.AUTO_BATCH`
  Esta constante indica que la ejecución implícita se produce cuando el número de sentencias del lote alcanza un valor definido por SQLJ.

  Entero positivo
  Cuando el número de sentencias añadidas al lote alcanza este valor, SQLJ ejecuta el lote implícitamente. Sin embargo, el proceso por lotes debería ejecutarse antes de que estas muchas sentencias se hayan añadido en el caso de que SQLJ encuentre una sentencia que se pueda procesar por lotes pero que sea incompatible o bien que la sentencia no se pueda procesar por lotes.

  Para determinar el número de filas que fueron actualizadas por las sentencias de un lote que se ejecutó implícitamente, invoque el método `ExecutionContext.getBatchUpdateCounts`. `getBatchUpdateCounts` devuelve una matriz entera que contiene el número de filas que fueron actualizadas por cada sentencia del lote. El orden de los elementos de la matriz corresponde al orden en el que se añadieron las sentencias al lote. Cada elemento de la matriz puede ser uno de los valores siguientes:

  -2 Este valor indica que la sentencia de SQL se ejecutó satisfactoriamente, pero no se pudo determinar el número de filas que fueron actualizadas.

  -3 Este valor indica que la sentencia de SQL falló.

  Otro valor entero
  Este valor es el número de filas que fueron actualizadas por la sentencia.

6. Opcionalmente, una vez añadidas todas las sentencias al lote, puede inhabilitar el proceso por lotes.

   Para ello invoque el método `ExecutionContext.setBatching(false)`. Cuando inhabilita el proceso por lotes, puede todavía ejecutar el lote implícita o explícitamente, pero no se añaden más sentencias al lote. Inhabilitar el proceso por lotes es útil cuando ya existe un lote y desea ejecutar una sentencia compatible de proceso por lotes, en lugar de añadirla al lote.

   Si desea eliminar un lote sin ejecutarlo, invoque el método `ExecutionContext.cancel`.

7. Si la ejecución por lotes era implícita, realice una ejecución final, explícita de `executeBatch` para asegurarse de que se hayan ejecutado todas las sentencias.

Ejemplo de actualización por lotes: el fragmento de código siguiente asigna aumentos de salario a todos los directores de departamento mediante la ejecución

de actualizaciones por lotes. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
#sql iterator GetMgr(String);  // Declarar iterador de posición
{
  GetMgr deptiter;            // Declarar objeto de la clase GetMgr
  String mgrnum = null;       // Declarar variable de lenguaje principal
                              // para número de director de departamento
  int raise = 400®;     // Declarar importe del aumento de salario
  int currentSalary;          // Declarar salario actual
  String url, username, password;   // Declarar url, ID de usuario, contraseña
  ...
  TestContext c1 = new TestContext (url, username, password, false);  1
  ExecutionContext ec = new ExecutionContext();                       2
  ec.setBatching(true);                                               3

  #sql [c1] deptiter =
    {SELECT MGRNO FROM DEPARTMENT};
                         // Asignar tabla de resultados de SELECT
                         // al objeto de iterador deptiter
  #sql {FETCH :deptiter INTO :mgrnum};
                        // Obtener el número del primer director de departamento
  while (!deptiter.endFetch()) {    // Determinar si FETCH devolvió una fila
    #sql [c1]
      {SELECT SALARY INTO :currentSalary FROM EMPLOYEE
        WHERE EMPNO=:mgrnum};
    #sql [c1, ec]                                    4
      {UPDATE EMPLOYEE SET SALARY=:(currentSalary+raise)
        WHERE EMPNO=:mgrnum};
    #sql {FETCH :deptiter INTO :mgrnum };
                            // Obtener la fila siguiente
  }
  ec.executeBatch();                                                  5
  ec.setBatching(false);                                              6
  #sql [c1] {COMMIT};
  deptiter.close();              // Cerrar el iterador
  ec.close();                    // Cerrar el contexto de ejecución
  c1.close();                    // Cerrar la conexión
}
```

Figura 55. Realización de una actualización por lotes

Cuando se produce un error durante la ejecución de una sentencia de un lote, se ejecutan las sentencias y se emite una excepción `BatchUpdateException` una vez ejecutadas todas las sentencias del lote. Consulte el tema Realización de actualizaciones por lotes en una aplicación JDBC para conocer cómo procesar una excepción `BatchUpdateException`.

Para obtener información sobre avisos, utilice el método `Statement.getWarnings` para el objeto en el que ejecutó el método `executeBatch`. Luego puede obtener una descripción de error, el estado de SQL y el código de error correspondientes a cada objeto `SQLWarning`.

Cuando un lote se ejecuta implícitamente debido a que el programa contiene una sentencia que no se puede añadir al lote, el lote se ejecuta antes de procesar la nueva sentencia. Si se produce un error al ejecutar el lote, la sentencia que provocó la ejecución del lote no se ejecuta.

Recomendación: desactiva la confirmación automática (autocommit) cuando realice actualizaciones por lotes para que pueda controlar si se deben confirmar los cambios en sentencias ya ejecutadas cuando se produzca un error durante la ejecución del lote.

Tareas relacionadas:

- "Realización de actualizaciones por lotes en aplicaciones JDBC" en la página 330
- "Conexión a una fuente de datos utilizando SQLJ" en la página 349
- "Control de la ejecución de sentencias de SQL en SQLJ" en la página 381

Información relacionada:

- "Clases e interfaces selectas de sqlj.runtime" en la página 438

Uso de iteradores pasados como variables en operaciones UPDATE o DELETE de posición de una aplicación SQLJ

SQLJ permite pasar iteradores entre métodos en calidad de variables. Un iterador que se utilice para una operación UPDATE o DELETE de posición solo se puede definir durante la ejecución. La misma sentencia UPDATE o DELETE de posición de SQLJ se puede utilizar con iteradores diferentes durante la ejecución. Si especifica el valor YES para -staticpositioned cuando personaliza su aplicación SQLJ como parte del proceso de preparación del programa, el personalizador de SQLJ prepara sentencias UPDATE o DELETE de posición para su ejecución estática. . En este caso, el personalizador debe determinar qué iteradores pertenecen a cada sentencia UPDATE o DELETE de posición. Para ello, el personalizador de SQLJ asocia tipos de datos de iterador con tipos de datos de las sentencias UPDATE o DELETE. Sin embargo, no existe una correspondencia unívoca entre las tablas de las sentencias UPDATE o DELETE y las clases de iterador, el personalizador de SQLJ no puede determinar exactamente qué iterador pertenece a cada sentencia UPDATE o DELETE. En este caso, el personalizador de SQLJ debe asociar arbitrariamente iteradores con sentencias UPDATE o DELETE, lo cual puede a veces producir errores de SQL. Esto se muestra en los fragmentos de código siguientes.

```
#sql iterator GeneralIter ( String );

  public static void main ( String args[] )
  {
...
    GeneralIter iter1 = null;
    #sql [ctxt] iter1 = { SELECT CHAR_COL1 FROM TABLE1 };

    GeneralIter iter2 = null;
    #sql [ctxt] iter2 = { SELECT CHAR_COL2 FROM TABLE2 };
...

    doUpdate ( iter1 );
  }

  public static void doUpdate ( GeneralIter iter )
  {
    #sql [ctxt] { UPDATE TABLE1 ... WHERE CURRENT OF :iter };
  }
```

Figura 56. Sentencia UPDATE estática de posición que se ejecuta satisfactoriamente

Este ejemplo define un solo iterador. Se crean dos instancias de ese iterador, y cada una de ellas se asocia a una sentencia SELECT diferente que recupera datos de una tabla diferente. Debido a que el iterador se pasa como variable al método `doUpdate`, no se puede conocer cuál de las instancias de iterador se utiliza para el UPDATE de posición hasta el momento de la ejecución. El proceso de vinculación de DB2® utiliza la primera instancia de iterador, `iter1`, cuando vincula el plan DB2. Durante

la ejecución, si iter1 se pasa al método doUpdate, tal como muestra la Figura 56 en la página 387, la operación UPDATE se ejecuta satisfactoriamente, pues iter1 y la sentencia UPDATE utilizan ambos TABLE1.

Si el programa está escrito de forma ligeramente diferente, tal como muestra la Figura 57, el proceso de vinculación de DB2 falla, aunque el programa parezca ser válido.

```
#sql iterator GeneralIter ( String );

  public static void main ( String args[] )
  {
...
    GeneralIter iter2 = null;
    #sql [ctxt] iter2 = { SELECT CHAR_COL2 FROM TABLE2 };
    GeneralIter iter1 = null;
    #sql [ctxt] iter1 = { SELECT CHAR_COL1 FROM TABLE1 };

...

    doUpdate ( iter1 );
  }

  public static void doUpdate ( GeneralIter iter )
  {
    #sql [ctxt] { UPDATE TABLE1 ... WHERE CURRENT OF :iter };
  }
```

Figura 57. Sentencia UPDATE estática de posición que falla durante la vinculación

En este caso, el proceso de vinculación de DB2 asocia iter2 con la sentencia UPDATE de posición, pues iter2 aparece en primer lugar en el programa. Cuando DB2 vincula el plan para el programa, la vinculación falla y devuelve el código de SQL -509, pues iter2 utiliza TABLE2 y la sentencia UPDATE utiliza TABLE1. Sin embargo, si se permite que el programa se vincule correctamente y se pasa iter1 al método doUpdate, el programa se ejecuta satisfactoriamente.

Puede evitar un error de vinculación para un programa como el de la Figura 57 especificando la opción de vinculación SQLERROR(CONTINUE) de DB2. Sin embargo, esta técnica tiene el inconveniente de que hace que DB2 cree un paquete, con independencia de los errores de SQL que existan en el programa. Una técnica mejor consiste en escribir el programa de forma que exista una correspondencia unívoca entre las tablas de las sentencias UPDATE o DELETE de posición y las clases de iterador. La Figura 58 en la página 389 muestra un ejemplo de cómo hacer esto.

```
#sql iterator Table2Iter(String);
#sql iterator Table1Iter(String);
  public static void main ( String args[] )
  {
...
    Table2Iter iter2 = null;
    #sql [ctxt] iter2 = { SELECT CHAR_COL2 FROM TABLE2 };

    Table1Iter iter1 = null;
    #sql [ctxt] iter1 = { SELECT CHAR_COL1 FROM TABLE1 };
...

    doUpdate(iter1);

  }

 public static void doUpdate ( Table1Iter iter )
  {
    ...
    #sql [ctxt] { UPDATE TABLE1 ... WHERE CURRENT OF :iter };
    ...
  }
 public static void doUpdate ( Table2Iter iter )
  {
    ...
    #sql [ctxt] { UPDATE TABLE2 ... WHERE CURRENT OF :iter };
    ...
  }
```

Figura 58. Sentencia UPDATE estática de posición que se ejecuta satisfactoriamente con independencia del orden de los iteradores

Con esta forma de codificación, cada clase de iterador se asocia a una sola tabla. Por tanto, el proceso de vinculación de DB2 puede siempre asociar la sentencia UPDATE de posición con un iterador válido.

Tareas relacionadas:

- "Ejecución de operaciones UPDATE y DELETE de posición en una aplicación SQLJ" en la página 363

Información relacionada:

- "db2sqljcustomize - Mandato Personalizador de perfiles SQLJ de DB2" en la publicación *Consulta de mandatos*

Utilización de iteradores desplazables en una aplicación SQLJ

Además de avanzar fila a fila por una tabla de resultados, puede desear ir hacia atrás o directamente a una fila determinada. El Controlador JDBC universal de DB2 proporciona esta capacidad.

Un iterador en el que se puede desplazar hacia delante, hacia atrás o hasta una fila determinada se denomina *iterador desplazable*. Un iterador desplazable de SQLJ es equivalente a la tabla de resultados de un cursor DB2® que esté declarado como SCROLL.

Al igual que un cursor desplazable, un iterador desplazable puede ser *sensible* o *insensible*. Un iterador desplazable sensible puede ser *estático* o *dinámico*. Insensible significa que los cambios hechos en la tabla subyacente después de abrir el iterador no son visibles para el iterador. Los iteradores insensibles son de solo lectura.

Sensible significa que los cambios que el iterador u otros procesos realizan en la tabla subyacente son visibles para el iterador.

Si un iterador desplazable es estático, el tamaño de la tabla de resultados y el orden de las filas en la tabla de resultados no cambian después de abrir el iterador. Esto significa que no puede insertar datos en tablas de resultados, y si suprime una fila de una tabla de resultados, se produce un hueco por supresión. Si actualiza una fila de la tabla de resultados y como consecuencia la fila deja de ser apropiada para la tabla de resultados, se produce un hueco por actualización. Una operación de recuperación de datos realizada en un hueco produce una `excepción de SQL`.

Si un iterador desplazable es dinámico, el tamaño de la tabla de resultados y el orden de las filas en la tabla de resultados pueden cambiar después de abrir el iterador. Las filas que se insertan o suprimen con sentencias INSERT y DELETE ejecutadas por el mismo proceso de aplicación son visibles inmediatamente. Las filas que se insertan o suprimen con sentencias INSERT y DELETE ejecutadas por otros procesos de aplicación son visibles una vez confirmados los cambios.

Para crear y utilizar un iterador desplazable, debe seguir estos pasos:

1. Especifique una cláusula de declaración de iterador que incluya las cláusulas siguientes:
   - `implements sqlj.runtime.Scrollable`

     Esto indica que el iterador es desplazable.
   - `with (sensitivity=INSENSITIVE|SENSITIVE)` o `with (sensitivity=SENSITIVE, dynamic=true|false)`

     `sensitivity=INSENSITIVE|SENSITIVE` indica si las operaciones de actualización o supresión realizadas sobre la tabla subyacente pueden ser visibles para el iterador. El valor por omisión del atributo "sensitivity" es `INSENSITIVE`.

     `dynamic=true|false` indica si el tamaño de la tabla de resultados o el orden de las filas en la tabla puede cambiar después de abrir el iterador. El valor por omisión del atributo "dynamic" es `false`.

   El iterador puede ser un iterador de nombre o de posición. Por ejemplo, la siguiente cláusula de declaración de iterador declara un iterador de posición que es sensible, dinámico y desplazable:

   ```
   #sql public iterator ByPos
     implements sqlj.runtime.Scrollable
     with (sensitivity=SENSITIVE, dynamic=true)  (String);
   ```

   La siguiente cláusula de declaración de iterador declara un iterador de nombre que es insensible y desplazable:

   ```
   #sql public iterator ByName
     implements sqlj.runtime.Scrollable
     with (sensitivity=INSENSITIVE) (String EmpNo);
   ```

   Restricción: no puede utilizar un iterador desplazable para seleccionar columnas de una tabla en un servidor de DB2 UDB para Linux, UNIX y Windows que tengan los tipos de datos siguientes:
   - LONG VARCHAR
   - LONG VARGRAPHIC
   - DATALINK
   - BLOB
   - CLOB

- Un tipo diferenciado que esté basado en cualquiera de los tipos de datos anteriores de esta lista
- Un tipo estructurado

2. Cree un objeto de iterador, que es una instancia de la clase de iterador utilizada.
3. Si desea indicar al entorno de ejecución de SQLJ la dirección inicial de la recuperación de datos, utilice el método `setFetchDirection(int dirección)`. *dirección* puede ser `FETCH_FORWARD` o `FETCH_REVERSE`. Si no invoca `setFetchDirection`, la dirección de recuperación de datos es `FETCH_FORWARD`.
4. Para cada fila a la que desee acceder:
   - En el caso de un iterador de nombre, siga estos pasos:
     a. Posicione el cursor utilizando uno de los métodos listados en la Tabla 38.

Tabla 38. Métodos sqlj.runtime.Scrollable para posicionar un cursor desplazable

Método	Posiciona el cursor
`first()`	En la primera fila de la tabla de resultados
`last()`	En la última fila de la tabla de resultados
`previous()`[1]	En la fila anterior de la tabla de resultados
`next()`	En la fila siguiente de la tabla de resultados
`absolute(int n)`[2]	Si $n>0$, en la fila n de la tabla de resultados. Si $n<0$ y m es el número de filas de la tabla de resultados, sitúa el iterador en la fila $m+n+1$ de la tabla de resultados.
`relative(int n)`[3]	Si $n>0$, en la fila que está situada n filas después de la fila actual. Si $n<0$, en la fila que está situada n filas antes de la fila actual. Si $n=0$, en la fila actual.
`afterLast()`	Después de la última fila de la tabla de resultados
`beforeFirst()`	Antes de la primera fila de la tabla de resultados

Notas:

1. Si el cursor está situado después de la última fila de la tabla de resultados, este método posiciona el cursor en la última fila.
2. Si el valor absoluto de n es mayor que el número de filas de la tabla de resultados, este método posiciona el cursor después de la última fila si n es positivo, o antes de la primera fila si n es negativo.
3. Suponga que m es el número de filas de la tabla de resultados y x es el número de fila actual de la tabla de resultados. Si $n>0$ y $x+n>m$, el iterador se sitúa después de la última fila. Si $n<0$ y $x+n<1$, el iterador se sitúa antes de la primera fila.

     b. Si necesita conocer la posición actual del cursor, utilice el método `getRow`, `isFirst`, `isLast`, `isBeforeFirst` o `isAfterLast` para obtener esa información. Si necesita conocer la dirección actual de recuperación de datos, invoque el método `getFetchDirection`.
     c. Utilice métodos accesores para recuperar la fila actual de la tabla de resultados.
     d. Si las operaciones de actualización o supresión realizadas por el iterador o por otros medios son visibles en la tabla de resultados, invoque el método `getWarnings` para determinar si la fila actual es un hueco.
   - En el caso de un iterador de posición, siga estos pasos:
     a. Utilice una sentencia FETCH con una cláusula de dirección de la recuperación de datos para posicionar el iterador y recuperar la fila actual de la tabla de resultados. La Tabla 39 en la página 392 lista las cláusulas que puede utilizar para posicionar el cursor.

Tabla 39. Cláusulas de FETCH para posicionar un cursor desplazable

Método	Posiciona el cursor
FIRST	En la primera fila de la tabla de resultados
LAST	En la última fila de la tabla de resultados
PRIOR[1]	En la fila anterior de la tabla de resultados
NEXT	En la fila siguiente de la tabla de resultados
ABSOLUTE(*n*)[2]	Si *n*>0, en la fila *n* de la tabla de resultados. Si *n*<0 y *m* es el número de filas de la tabla de resultados, sitúa el iterador en la fila *m+n+1* de la tabla de resultados.
RELATIVE(*n*)[3]	Si *n*>0, en la fila que está situada *n* filas después de la fila actual. Si *n*<0, en la fila que está situada *n* filas antes de la fila actual. Si *n*=0, en la fila actual.
AFTER[4]	Después de la última fila de la tabla de resultados
BEFORE[4]	Antes de la primera fila de la tabla de resultados

Notas:

1. Si el cursor está situado después de la última fila de la tabla de resultados, este método posiciona el cursor en la última fila.
2. Si el valor absoluto de *n* es mayor que el número de filas de la tabla de resultados, este método posiciona el cursor después de la última fila si *n* es positivo, o antes de la primera fila si *n* es negativo.
3. Suponga que *m* es el número de filas de la tabla de resultados y *x* es el número de fila actual de la tabla de resultados. Si *n*>0 y *x+n*>*m*, el iterador se sitúa después de la última fila. Si *n*<0 y *x+n*<1, el iterador se sitúa antes de la primera fila.
4. No se asignan valores a las expresiones de lenguaje principal.

   b. Si las operaciones de actualización o supresión realizadas por el iterador o por otros medios son visibles en la tabla de resultados, invoque el método `getWarnings` para determinar si la fila actual es un hueco.

5. Invoque el método `close` para cerrar el iterador.

Por ejemplo, el código siguiente muestra cómo utilizar un iterador de nombre para recuperar el número de empleado y apellido en todas las filas de la tabla EMPLOYEE en orden inverso. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
#sql  iterator ScrollIter implements sqlj.runtime.Scrollable   1
  (String EmpNo, String LastName);
{
  ScrollIter scrliter;                                          2
  #sql [ctxt]
  scrliter={SELECT EMPNO, LASTNAME FROM EMPLOYEE};
  scrliter.afterLast();
  while (scrliter.previous())                                   4a
  {
    System.out.println(scrliter.EmpNo() + " "                   4c
      + scrliter.LastName());
  }
  scrliter.close();                                             5
}
```

Figura 59. Utilización de iteradores desplazables

Conceptos relacionados:

- "Recuperación de datos de tablas DB2 por una aplicación SQLJ" en la página 357

Tareas relacionadas:

- "Utilización de un iterador de nombre en una aplicación SQLJ" en la página 358
- "Utilización de un iterador de posición en una aplicación SQLJ" en la página 361

Capítulo 17. Información de consulta sobre JDBC y SQLJ

Las secciones siguientes contienen información de consulta sobre métodos de JDBC y cláusulas de SQLJ.

Tipos de datos de Java, JDBC y SQL

Las tablas siguientes resumen las correlaciones de tipos de datos de Java con tipos de datos de SQL para un sistema DB2 UDB para Linux, UNIX y Windows.

La Tabla 40 resume las correlaciones de tipos de datos de Java con tipos de datos de DB2 para métodos `PreparedStatement.setXXX` o `ResultSet.updateXXX` de programas JDBC, y para expresiones de lenguaje principal de entrada contenidas en programas SQLJ. Cuando aparece listado más de un tipo de datos, el primer tipo de datos es el recomendado.

Tabla 40. Correlaciones de tipos de datos de Java con tipos de datos de DB2 para actualizar tablas DB2

Tipo de datos de Java	Tipo de datos de SQL
short, boolean[1], byte[1]	SMALLINT
int, java.lang.Integer	INTEGER
long, java.lang.Long	DECIMAL(19,0)[2]
long, java.lang.Long	BIGINT[3]
float, java.lang.Float	REAL
double, java.lang.Double	DOUBLE
java.math.BigDecimal	DECIMAL(*p*,*s*)[4]
java.lang.String	CHAR(*n*)[5]
java.lang.String	GRAPHIC(*m*)[6]
java.lang.String	VARCHAR(*n*)[7]
java.lang.String	VARGRAPHIC(*m*)[8]
java.lang.String	CLOB(*n*)[9]
byte[]	CHAR(*n*) FOR BIT DATA[5]
byte[]	VARCHAR(*n*) FOR BIT DATA[7]
byte[]	BLOB(*n*)[9,10]
byte[]	ROWID
java.sql.Blob	BLOB(*n*)[10]
java.sql.Clob	CLOB(*n*)[10]
java.sql.Clob	DBCLOB(*m*)[11]
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP
java.io.ByteArrayInputStream	BLOB(*n*)[10]
java.io.StringReader	CLOB(*n*)[10]
java.io.ByteArrayInputStream	CLOB(*n*)[10]

Tabla 40. Correlaciones de tipos de datos de Java con tipos de datos de DB2 para actualizar tablas DB2 (continuación)

Tipo de datos de Java	Tipo de datos de SQL
com.ibm.db2.jcc.DB2RowID	ROWID
java.net.URL	DATALINK[12]

Notas:

1. DB2 no tiene un equivalente exacto para los tipos de datos boolean o byte de Java, pero la mejor correspondencia es SMALLINT.
2. DB2 UDB en el entorno OS/390 o z/OS no tiene un equivalente exacto para los tipos de datos long o java.lang.Long de Java, pero la mejor correspondencia es DECIMAL(19,0).
3. El tipo de datos BIGINT de SQL solo está disponible en DB2 UDB para Linux, UNIX y Windows.
4. *p* es la precisión decimal y *s* es la escala de la columna DB2.

   Es conveniente que diseñe las aplicaciones financieras de forma que las columnas de tipo java.math.BigDecimal se correlacionen con columnas de tipo DECIMAL. Si conoce la precisión y escala de una columna DECIMAL, la actualización de datos de esa columna con datos de una variable java.math.BigDecimal produce una mejor precisión y resultados que el uso de cualquier otra combinación de tipos de datos.
5. *n*<=254.
6. *m*<=127.
7. *n*<=32672.
8. *m*<=16336.
9. Esta correlación solo es válida si DB2 puede determinar el tipo de datos de la columna.
10. *n*<=2147483647.
11. *m*<=1073741823.
12. El tipo de datos DATALINK solo está soportado por el Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows.

La Tabla 41 resume las correlaciones de tipos de datos de DB2 con tipos de datos de Java para métodos `ResultSet.getXXX` de programas JDBC y para iteradores de programas SQLJ. Esta tabla no lista los tipos de objeto de reiniciador numérico de Java, que se recuperan utilizando `ResultSet.getObject`.

Tabla 41. Correlaciones de tipos de datos de DB2 con tipos de datos de Java para recuperar datos de tablas DB2

Tipo de datos de SQL	Tipo de datos de Java o tipo de objeto de Java recomendado	Otros tipos de datos de Java soportados
SMALLINT	short	byte, int, long, float, double, java.math.BigDecimal, boolean, java.lang.String
INTEGER	int	short, byte, long, float, double, java.math.BigDecimal, boolean, java.lang.String
BIGINT[1]	long	int, short, byte, float, double, java.math.BigDecimal, boolean, java.lang.String
REAL	float	long, int, short, byte, double, java.math.BigDecimal, boolean, java.lang.String
DOUBLE	double	long, int, short, byte, float, java.math.BigDecimal, boolean, java.lang.String

Tabla 41. Correlaciones de tipos de datos de DB2 con tipos de datos de Java para recuperar datos de tablas DB2 (continuación)

Tipo de datos de SQL	Tipo de datos de Java o tipo de objeto de Java recomendado	Otros tipos de datos de Java soportados
CHAR(*n*)	java.lang.String	long, int, short, byte, float, double, java.math.BigDecimal, boolean, java.sql.Date, java.sql.Time, java.sql.Timestamp, java.io.InputStream, java.io.Reader
VARCHAR(*n*)	java.lang.String	long, int, short, byte, float, double, java.math.BigDecimal, boolean, java.sql.Date, java.sql.Time, java.sql.Timestamp, java.io.InputStream, java.io.Reader
CHAR(*n*) FOR BIT DATA	byte[]	java.lang.String, java.io.InputStream, java.io.Reader
VARCHAR(*n*) FOR BIT DATA	byte[]	java.lang.String, java.io.InputStream, java.io.Reader
GRAPHIC(*m*)	java.lang.String	long, int, short, byte, float, double, java.math.BigDecimal, boolean, java.sql.Date, java.sql.Time, java.sql.Timestamp, java.io.InputStream, java.io.Reader
VARGRAPHIC(*m*)	java.lang.String	long, int, short, byte, float, double, java.math.BigDecimal, boolean, java.sql.Date, java.sql.Time, java.sql.Timestamp, java.io.InputStream, java.io.Reader
CLOB(*n*)	java.sql.Clob	java.lang.String
BLOB(*n*)	java.sql.Blob	byte[][3]
DBCLOB(*m*)	Sin equivalente exacto. Utilice java.sql.Clob.	
ROWID	com.ibm.db2.jcc.DB2RowID	byte[]
DATE	java.sql.Date	java.sql.String, java.sql.Timestamp
TIME	java.sql.Time	java.sql.String, java.sql.Timestamp
TIMESTAMP	java.sql.Timestamp	java.sql.String, java.sql.Date, java.sql.Time, java.sql.Timestamp
DATALINK	java.net.URL[4]	

Notas:

1. El tipo de datos BIGINT de SQL solo está disponible en DB2 UDB para Linux, UNIX y Windows.
2. Es conveniente que diseñe las aplicaciones financieras de forma que las columnas de tipo DECIMAL se correlacionen con columnas de tipo java.math.BigDecimal. Si conoce la precisión y escala de una columna DECIMAL, la recuperación de datos de esa columna para colocarlos en una variable java.math.BigDecimal produce una mejor precisión y resultados que el uso de cualquier otra combinación de tipos de datos.
3. Esta correlación solo es válida si DB2 puede determinar el tipo de datos de la columna.
4. El tipo de datos DATALINK solo está soportado por el Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows.

La Tabla 42 en la página 398 resume las correlaciones de tipos de datos de Java con tipos de datos de JDBC y DB2 para parámetros de una función definida por el usuario y de un procedimiento almacenado. Las correlaciones de tipos de datos de

Java con tipos de datos de JDBC son para métodos `CallableStatement.registerOutParameter` de programas JDBC. Las correlaciones de tipos de datos de Java con tipos de datos de DB2 son para parámetros utilizados al invocar un procedimiento almacenado o función definida por el usuario.

Cuando en la Tabla 42 aparece listado más de un tipo de datos de Java, el primer tipo de datos es el **recomendado**.

Tabla 42. Correlaciones de tipos de datos de Java, JDBC y SQL para procedimientos almacenados y funciones definidas por el usuario

Tipo de datos de Java	Tipo de datos de JDBC	Tipo de datos de SQL
boolean[1]	BIT	SMALLINT
byte[1]	TINYINT	SMALLINT
short, java.lang.Integer	SMALLINT	SMALLINT
int, java.lang.Integer	INTEGER	INTEGER
long	BIGINT	BIGINT[2]
float, java.lang.Float	REAL	REAL
float, java.lang.Float	FLOAT	REAL
double, java.lang.Double	DOUBLE	DOUBLE
java.math.BigDecimal	NUMERIC	DECIMAL
java.math.BigDecimal	DECIMAL	DECIMAL
java.lang.String	CHAR	CHAR
java.lang.String	CHAR	GRAPHIC
java.lang.String	VARCHAR	VARCHAR
java.lang.String	VARCHAR	VARGRAPHIC
java.lang.String	LONGVARCHAR	VARCHAR
java.lang.String	VARCHAR	CLOB(*n*)
java.lang.String	LONGVARCHAR	CLOB(*n*)
java.lang.String	CLOB	CLOB(*n*)
byte[]	BINARY	CHAR FOR BIT DATA
byte[]	VARBINARY	VARCHAR FOR BIT DATA
byte[]	LONGVARBINARY	VARCHAR FOR BIT DATA
byte[]	VARBINARY	BLOB(*n*)[3]
byte[]	LONGVARBINARY	BLOB(*n*)[3]
java.sql.Date	DATE	DATE
java.sql.Time	TIME	TIME
java.sql.Timestamp	TIMESTAMP	TIMESTAMP
java.sql.Blob	BLOB	BLOB
java.sql.Clob	CLOB	CLOB
java.sql.Clob	CLOB	DBCLOB
java.io.ByteArrayInputStream	Ninguno	BLOB(*n*)
java.io.StringReader	Ninguno	CLOB(*n*)
java.io.ByteArrayInputStream	Ninguno	CLOB(*n*)
com.ibm.db2.jcc.DB2RowID	com.ibm.db2.jcc.DB2Types.ROWID	ROWID

Tabla 42. Correlaciones de tipos de datos de Java, JDBC y SQL para procedimientos almacenados y funciones definidas por el usuario (continuación)

Tipo de datos de Java	Tipo de datos de JDBC	Tipo de datos de SQL
java.net.URL	DATALINK	DATALINK[4]

Notas:

1. Un procedimiento almacenado o función definida por el usuario que esté definido con un parámetro de tipo SMALLINT se puede invocar con un parámetro de tipo boolean o byte. Sin embargo, esto no es recomendable.
2. El tipo de datos BIGINT de SQL solo está disponible en servidores DB2 UDB para Linux, UNIX y Windows.Para las aplicaciones Java que conectan desde un cliente DB2 UDB Versión 8.1 a un servidor DB2 UDB Versión 7, cuando se utiliza el método `CallableStatement.getObject` para recuperar un valor BIGINT, se devuelve un objeto `java.math.BigDecimal`.
3. Esta correlación solo es válida si DB2 puede determinar el tipo de datos de la columna.
4. El tipo de datos DATALINK solo está soportado por el Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows.

La Tabla 43 resume las correlaciones de tipos de datos para parámetros de SQL de una sentencia CREATE PROCEDURE o CREATE FUNCTION con los tipos de datos del correspondiente método de procedimiento almacenado o función definida por el usuario de Java.

Para DB2 UDB para Linux, UNIX y Windows, si aparece listado más de un tipo de datos de Java para un tipo de datos de SQL, solo es válido el **primer** tipo de datos de Java.

Para DB2 UDB en el entorno OS/390 o z/OS, si aparece listado más de un tipo de datos de Java, y utiliza como parámetro de método un tipo de datos que no sea el primero, es necesario incluir una signatura de método en la cláusula EXTERNAL de la sentencia CREATE PROCEDURE o CREATE FUNCTION que especifica los tipos de datos de Java de los parámetros del método.

Tabla 43. Correlaciones de tipos de datos de SQL de una sentencia CREATE PROCEDURE o CREATE FUNCTION con los tipos de datos del correspondiente programa de procedimiento almacenado o función definida por el usuario de Java

Tipo de datos de SQL en CREATE PROCEDURE o CREATE FUNCTION	Tipo de datos en el método de procedimiento almacenado o función definida por el usuario de Java
SMALLINT	short, java.lang.Integer
INTEGER	int, java.lang.Integer
BIGINT[1]	long
REAL	float, java.lang.Float
DOUBLE	double, java.lang.Double
DECIMAL	java.math.BigDecimal
CHAR	java.lang.String
GRAPHIC	java.lang.String
VARCHAR	java.lang.String
VARGRAPHIC	java.lang.String
CHAR FOR BIT DATA	byte[]
VARCHAR FOR BIT DATA	byte[]
DATE	java.sql.Date

Tabla 43. Correlaciones de tipos de datos de SQL de una sentencia CREATE PROCEDURE o CREATE FUNCTION con los tipos de datos del correspondiente programa de procedimiento almacenado o función definida por el usuario de Java (continuación)

Tipo de datos de SQL en CREATE PROCEDURE o CREATE FUNCTION	Tipo de datos en el método de procedimiento almacenado o función definida por el usuario de Java
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp
BLOB	java.sql.Blob
CLOB	java.sql.Clob
DBCLOB	java.sql.Clob
ROWID	com.ibm.db2.jcc.DB2Types.ROWID
DATALINK	java.net.URL[2]

Notas:

1. El tipo de datos BIGINT de SQL solo está disponible en servidores DB2 UDB para Linux, UNIX y Windows.
2. El tipo de datos DATALINK solo está soportado por el Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows.

Información relacionada:

- "Diferencias de JDBC entre el Controlador JDBC de DB2 Universal y otros controladores JDBC de DB2" en la página 459

Propiedades del Controlador JDBC de DB2 Universal

Las propiedades definen cómo se debe establecer la conexión con una fuente de datos determinada. A menos que se indique otra cosa, las propiedades se pueden definir para un objeto `DataSource` o para un objeto `Connection`. Puede definir propiedades en una de las maneras siguientes:

- Utilizando métodos `setXXX`

  Las propiedades son aplicables a las siguientes implementaciones específicas de DB2 que las heredan de `com.ibm.db2.jcc.DB2BaseDataSource` :
  - `com.ibm.db2.jcc.DB2SimpleDataSource`
  - `com.ibm.db2.jcc.DB2DataSource`
  - `com.ibm.db2.jcc.DB2ConnectionPoolDataSource`
  - `com.ibm.db2.jcc.DB2XADataSource`

  Consulte el tema Resumen de las extensiones del Controlador JDBC de DB2 Universal para JDBC para ver un resumen de los nombres de propiedades y tipos de datos.
- En un valor de `java.util.Properties` en el parámetro *info* de una llamada `DriverManager.getConnection`, tal como se muestra en el tema Conectar con una fuente de datos utilizando la interfaz DriverManager con el Controlador JDBC de DB2 Universal.
- En un valor de `java.lang.String` en el parámetro *url* de una llamada DriverManager.getConnection, tal como se muestra en el tema Conectar con una fuente de datos utilizando la interfaz DriverManager con el Controlador JDBC de DB2 Universal.

Las propiedades son:

activeServerListJNDIName
Identifica una referencia de JNDI a una instancia de `DB2ActiveServerList`

contenida en un depósito JNDI de información sobre un servidor alternativo. Si el valor de activeServerListJNDIName no es nulo, las conexiones se pueden transferir a un servidor alternativo que está especificado en la instancia de `DB2ActiveServerList` referenciada por el valor. Si activeServerListJNDIName es nulo, la conexiones no se pueden transferir a otro servidor utilizando la información sobre un servidor alternativo contenida en un depósito JNDI.

clientAccountingInformation
Especifica información contable correspondiente al cliente actual de la conexión. Esta información se utiliza con fines de contabilidad de clientes. Este valor puede cambiar durante una conexión. El tipo de datos de esta propiedad es String. Para un servidor DB2 UDB para Linux, UNIX y Windows, la longitud máxima es 255 bytes. Es válido especificar una serie vacía Java ("") para este valor, pero no es válido utilizar un valor `nulo` Java.

clientApplicationInformation
Especifica información de la aplicación correspondiente al cliente actual de la conexión. Esta información se utiliza con fines de contabilidad de clientes. Este valor puede cambiar durante una conexión. El tipo de datos de esta propiedad es String. Para un servidor DB2 UDB para Linux, UNIX y Windows, la longitud máxima es 255 bytes. Es válido especificar una serie vacía Java ("") para este valor, pero no es válido utilizar un valor `nulo` Java.

clientUser
Especifica el nombre de usuario del cliente actual de la conexión. Esta información se utiliza con fines de contabilidad de clientes. A diferencia del nombre de usuario de una conexión JDBC, este valor puede cambiar durante una conexión. Para un servidor DB2 UDB para Linux, UNIX y Windows, la longitud máxima es 255 bytes.

clientWorkstation
Especifica el nombre de la estación de trabajo correspondiente al cliente actual de la conexión. Esta información se utiliza con fines de contabilidad de clientes. Este valor puede cambiar durante una conexión. El tipo de datos de esta propiedad es String. Para un servidor DB2 UDB para Linux, UNIX y Windows, la longitud máxima es 255 bytes. Es válido especificar una serie vacía Java ("") para este valor, pero no es válido utilizar un valor `nulo` Java.

cliSchema
Especifica el esquema de las tablas o vistas de catálogo transitorias de DB2 que se examinan cuando una aplicación invoca un método de `DatabaseMetaData`.

currentFunctionPath
Especifica la vía de SQL que se utiliza para resolver nombres de tipos de datos y de funciones no calificados en las sentencias de SQL que están contenidas en programas de JDBC. El tipo de datos de esta propiedad es String. Para un servidor DB2 UDB para Linux, UNIX y Windows, la longitud máxima es 254 bytes. El valor es una lista de nombres de esquema separados por comas. Esos nombres pueden ser identificadores ordinarios o delimitados.

currentLockTimeout
Da instrucciones a servidores DB2 UDB para Linux, UNIX y Windows para que estén a la espera indefinidamente a causa de un bloqueo o para que esperen durante un número especificado de segundos a causa de un bloqueo que no se puede obtener de inmediato. El tipo de datos de esta propiedad es int. Un valor igual a cero denota ausencia de espera. Un valor igual a -1 significa una espera indefinida. Un valor entero positivo indica el número de segundos que se debe esperar a causa de un bloqueo.

currentPackagePath
Especifica un lista separada por comas de colecciones contenidas en el servidor. El servidor DB2 examina estas colecciones para buscar los paquetes de DB2 del Controlador JDBC universal de DB2.

Las reglas de prioridad de las propiedades currentPackagePath y currentPackageSet se ajustan a las reglas de prioridad de los registros especiales DB2 CURRENT PACKAGESET y CURRENT PACKAGE PATH.

currentPackageSet
Especifica el ID de colección para buscar paquetes de DB2 para el Controlador JDBC universal de DB2. El tipo de datos de esta propiedad es String. El valor por omisión es NULLID. Si currentPackageSet está definido, su valor tiene prioridad sobre el valor de jdbcCollection.

Se pueden instalar varias instancias del Controlador JDBC universal de DB2 en un servidor de bases de datos ejecutando varias veces el programa de utilidad DB2binder. El programa de utilidad DB2binder incluye la opción -collection, que permite que el instalador especifique el ID de colección para cada instancia del Controlador JDBC universal de DB2. Para seleccionar una instancia del Controlador JDBC universal de DB2 para una conexión, especifique un valor de currentPackageSet que coincida con el ID de colección para una de las instancias del Controlador JDBC universal de DB2.

Las reglas de prioridad de las propiedades currentPackagePath y currentPackageSet se ajustan a las reglas de prioridad de los registros especiales DB2 CURRENT PACKAGESET y CURRENT PACKAGE PATH.

currentSchema
Especifica el nombre de esquema por omisión que se utiliza para calificar objetos de base de datos no calificados en sentencias de SQL preparadas dinámicamente. El valor de esta propiedad define el valor del registro especial CURRENT SCHEMA en un servidor que no sea un servidor DB2 UDB para z/OS. No defina esta propiedad para un servidor DB2 UDB para z/OS.

currentSQLID
Especifica:

- El ID de autorización que se utiliza para la comprobación de autorizaciones en las sentencias CREATE, GRANT y REVOKE de SQL preparadas dinámicamente.
- El propietario de un espacio de tablas, base de datos, grupo de almacenamiento o sinónimo que es creado por una sentencia CREATE emitida dinámicamente.
- El calificador implícito de todos los nombres de tabla, vista, alias e índice especificados en sentencias de SQL dinámico.

La propiedad currentSQLID define el valor del registro especial CURRENT SQLID en un servidor DB2 UDB para z/OS server. Si la propiedad currentSQLID no está definida, el nombre de esquema por omisión es el valor del registro especial CURRENT SQLID.

cursorSensitivity
Especifica si el valor de `java.sql.ResultSet.TYPE_SCROLL_SENSITIVE` para un `ResultSet` de JDBC se correlaciona con el atributo SENSITIVE DYNAMIC o SENSITIVE STATIC para el cursor DB2 asociado. Los valores posibles son `TYPE_SCROLL_SENSITIVE_STATIC` y `TYPE_SCROLL_SENSITIVE_DYNAMIC` . El valor por omisión es `TYPE_SCROLL_SENSITIVE_STATIC`.

Esta propiedad no se tiene en cuenta para servidores de bases de datos que no dan soporte a cursores desplazables dinámicos sensibles.

databaseName
Especifica el nombre del servidor de bases de datos. Este nombre constituye la porción del URL de conexión correspondiente a la *base de datos*. El nombre depende de si se utiliza Conectividad de tipo 4 universal o Conectividad de tipo 2 universal.

Para Conectividad de tipo 4 universal:

- Si la conexión es con un servidor DB2 para z/OS, el valor de databaseName es el nombre de ubicación de DB2 que se define durante la instalación. Todos los caracteres de este valor deben ser caracteres en mayúsculas. Puede determinar el nombre de ubicación ejecutando la sentencia de SQL siguiente en el servidor:

  ```
  SELECT CURRENT SERVER FROM SYSIBM.SYSDUMMY1;
  ```

- Si la conexión es con un servidor DB2 UDB para Linux, UNIX y Windows, el valor de databaseName es el nombre de base de datos que se define durante la instalación.
- Si la conexión es con un servidor IBM Cloudscape, el valor de databaseName es el nombre totalmente calificado del archivo donde reside la base de datos. Este nombre debe estar encerrado entre comillas dobles ("). Por ejemplo:

  ```
  "c:/databases/testdb"
  ```

Si esta propiedad no está definida, las conexiones se establecen con el sitio local.

Para Conectividad de tipo 2 universal:

- El valor de databaseName es el nombre de base de datos que se define durante la instalación, si el valor de la propiedad de conexión serverName es nulo. Si el valor de la propiedad serverName no es nulo, el valor de databaseName es un alias de base de datos.

deferPrepares
Especifica si se difieren las operaciones de preparación hasta el momento de la ejecución. El tipo de datos de esta propiedad es boolean. El valor por omisión es `true` para Conectividad de tipo 4 universal. Esta propiedad no es aplicable a Conectividad de tipo 2 universal.

El diferir las operaciones de preparación puede reducir los retardos de la red. Sin embargo, si difiere las operaciones de preparación, asegúrese de que los tipos de entrada de coinciden con los tipos de columna de la tabla DB2.

description
Descripción de la fuente de datos. El tipo de datos de esta propiedad es String.

driverType
Para la interfaz `DataSource`, determina qué controlador utilizar para las conexiones. El tipo de datos de esta propiedad es int. Los valores válidos son 2 o 4. El valor por omisión es 2.

fullyMaterializeLobData
Indica si el controlador recupera localizadores de LOB para operaciones FETCH. El tipo de datos de esta propiedad es boolean. Si el valor es `true`, los datos LOB se materializan por completo dentro del controlador JDBC cuando se recupera una fila. Si este valor es `false`, los datos LOB se canalizan. El controlador utiliza localizadores internamente para recuperar datos LOB en forma de bloques a medida que sea necesario. Es muy recomendable que establezca este valor en `false` cuando recupere datos LOB que contengan grandes volúmenes de datos. El valor por omisión es `true`.

Esta propiedad no tiene ningún efecto para parámetros de procedimiento almacenado ni datos LOB que se recuperan utilizando cursores desplazables. Los parámetros de procedimiento almacenado de LOB se materializan siempre. Se utilizan siempre localizadores de LOB para los datos que se recuperan utilizando cursores desplazables.

gssCredential
Para una fuente de datos que utilice la seguridad Kerberos, especifica una credencial delegada que se pasa desde otro principal. El tipo de datos de esta propiedad es org.ietf.jgss.GSSCredential. Las credenciales delegadas se utilizan en los entornos de varios niveles, tales como cuando un cliente se conecta a WebSphere Application Server, el cual a su vez se conecta a DB2. El valor de esta propiedad se obtiene del cliente, invocando el método `GSSContext.getDelegCred`. `GSSContext` forma parte de la API GSS (Generic Security Service) de IBM para Java. Si define esta propiedad, debe también definir las propiedades Mechanism y KerberosServerPrincipal.

Esta propiedad solo es aplicable a Conectividad de tipo 4 universal.

Para obtener más información sobre la utilización de la seguridad Kerberos con el Controlador JDBC universal de DB2, consulte el tema Utilización de la seguridad Kerberos con el Controlador JDBC de DB2 Universal.

jdbcCollection
Especifica el ID de colección de los paquetes que son utilizados por una instancia del Controlador JDBC universal de DB2 durante la ejecución. El tipo de datos de jdbcCollection es String. El valor por omisión es NULLID.

Esta propiedad se utiliza con la opción -collection de DB2Binder. El programa de utilidad DB2Binder debe haber vinculado previamente paquetes del Controlador JDBC universal de DB2 en el servidor utilizando un valor de -collection que coincida con el valor de jdbcCollection.

El valor de jdbcCollection no determina la colección que se utiliza para aplicaciones SQLJ. Para SQLJ, la colección está determinada por la opción -collection del personalizador de SQLJ.

jdbcCollection no es aplicable a Conectividad de tipo 2 universal en DB2 UDB para z/OS.

kerberosServerPrincipal
Para una fuente de datos que utilice la seguridad Kerberos, especifica el nombre que se utiliza para la fuente de datos cuando se registra con el Centro de distribución de claves (KCD) de Kerberos. El tipo de datos de esta propiedad es String.

Esta propiedad solo es aplicable a Conectividad de tipo 4 universal.

loginTimeout
Es la cantidad máxima de tiempo, en segundos, que se debe esperar para establecer conexión con una fuente de datos, o para realizar peticiones de SQL a esa fuente de datos. Una vez transcurrido el número de segundos especificado por loginTimeout, el controlador cierra la conexión con la fuente de datos. El tipo de datos de esta propiedad es int. El valor por omisión es 0, que es el valor por omisión del sistema para el tiempo de espera. Esta propiedad no es aplicable a Conectividad de tipo 2 universal en DB2 UDB para el entorno z/OS u OS/390.

logWriter
Es la corriente de salida donde se escriben todos los mensajes de registro de anotaciones y de rastreo correspondientes al objeto `DataSource`. El tipo de

datos de esta propiedad es java.io.PrinterWriter. El valor por omisión es null, que significa que no se generan datos de registro de anotaciones ni de rastreo para DataSource.

password
Es la contraseña que se debe utilizar para establecer conexiones. El tipo de datos de esta propiedad es String. Cuando utiliza la interfaz DataSource para establecer una conexión, puede alterar temporalmente el valor de esta propiedad invocando esta modalidad del método DataSource.getConnection:

```
getConnection(usuario, contraseña);
```

portNumber
Es el número de puerto donde el servidor DRDA® recibe las peticiones. El tipo de datos de esta propiedad es int.

readOnly
Especifica si la conexión es de solo lectura. El tipo de datos de esta propiedad es boolean. El valor por omisión es false.

resultSetHoldability
Especifica si los cursores permanecen abiertos después de una operación de confirmación. El tipo de datos de esta propiedad es int. Los valores válidos son com.ibm.db2.jcc.DB2BaseDataSource.HOLD_CURSORS_OVER_COMMIT o com.ibm.db2.jcc.DB2BaseDataSource.CLOSE_CURSORS_AT_COMMIT. Estos valores son los mismos que las constantes ResultSet.HOLD_CURSORS_OVER_COMMIT y ResultSet.CLOSE_CURSORS_AT_COMMIT que se definen en JDBC 3.0.

retrieveMessagesFromServerOnGetMessage
Especifica si las llamadas SQLException.getMessage de JDBC hacen que el Controlador JDBC universal de DB2 invoque un procedimiento almacenado de DB2 UDB para OS/390 o z/OS que obtiene el texto del mensaje de error. El tipo de datos de esta propiedad es boolean. El valor por omisión es false, que significa que el texto completo del mensaje no se devuelve al cliente. Una alternativa al uso de true como valor de esta propiedad, es utilizar el método DB2Sqlca.getMessage, exclusivo de DB2, en las aplicaciones. Ambas técnicas producen una llamada de procedimiento almacenado, la cual inicia una unidad de trabajo.

securityMechanism
Especifica el mecanismo de seguridad de DRDA. El tipo de datos de esta propiedad es int. Los valores posibles son:

CLEAR_TEXT_PASSWORD_SECURITY
ID de usuario y contraseña

USER_ONLY_SECURITY
ID de usuario solamente

ENCRYPTED_PASSWORD_SECURITY
ID de usuario, contraseña cifrada

ENCRYPTED_USER_AND_PASSWORD_SECURITY
ID de usuario cifrado y contraseña cifrada

KERBEROS_SECURITY
Kerberos

Si se especifica esta propiedad, el mecanismo de seguridad especificado es el único mecanismo utilizado. Si el mecanismo de seguridad no está soportado por la conexión, se emite una excepción.

Si no se especifica ningún valor para esta propiedad, el peticionario intenta conectar utilizando el mecanismo más seguro que sea posible. Si no se establece una conexión porque el servidor no soporta el mecanismo de seguridad especificado, el servidor devuelve una lista de opciones alternativas al peticionario. El peticionario prueba cada uno de esos mecanismos de seguridad hasta que se pueda establecer una conexión con uno de ellos. Si no existen opciones alternativas, o si fallan todas ellas, se emite una excepción.

serverName
Es el nombre de sistema principal o dirección TCP/IP de la fuente de datos. El tipo de datos de esta propiedad es String.

traceFile
Especifica el nombre del archivo en donde el Controlador JDBC universal de DB2 escribe información de rastreo. El tipo de datos de esta propiedad es `String`. La propiedad traceFile es una alternativa al uso de la propiedad logWriter para encaminar la corriente de datos de rastreo de salida hacia un archivo.

traceFileAppend
Especifica si deben añadir o sobrescribir datos en el archivo especificado por la propiedad traceFile. El tipo de datos de esta propiedad es boolean. El valor por omisión es `false`, que significa que se sobrescribe el archivo especificado por la propiedad traceFile.

traceLevel
Especifica qué se debe rastrear. El tipo de datos de esta propiedad es int.

Puede especificar uno o más de los valores de rastreo siguientes con la propiedad traceLevel:
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_NONE`
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_CONNECTION_CALLS`
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_STATEMENT_CALLS`
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_RESULT_SET_CALLS`
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DRIVER_CONFIGURATION`
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_CONNECTS`
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DRDA_FLOWS`
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_RESULT_SET_META_DATA`
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_PARAMETER_META_DATA`
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DIAGNOSTICS`
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_SQLJ`
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_XA_CALLS (solamente en Conectividad de tipo 2 universal para DB2 UDB para Linux, UNIX y Windows)`
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL`

Para especificar más de un valor de rastreo, utilice una de estas técnicas:
- Utilice operadores OR (`|`) de bits con dos o más valores de rastreo. Por ejemplo, para rastrear flujos de DRDA y llamadas de conexión, especifique este valor para traceLevel:

  ```
  TRACE_DRDA_FLOWS|TRACE_CONNECTION_CALLS
  ```
- Utilice un operador de complemento a nivel de bit (~) con un valor de rastreo para especificar todos los rastreos excepto uno determinado. Por ejemplo, para rastrear todo excepto los flujos de DRDA, especifique este valor para traceLevel:

  ```
  ~TRACE_DRDA_FLOWS
  ```

user

Es el ID de usuario que se debe utilizar para establecer conexiones. El tipo de datos de esta propiedad es String. Cuando utiliza la interfaz `DataSource` para establecer una conexión, puede alterar temporalmente el valor de esta propiedad invocando esta modalidad del método `DataSource.getConnection`:

```
getConnection(usuario, contraseña);
```

Conceptos relacionados:

- "Seguridad cuando se utiliza el Controlador JDBC de DB2 Universal" en la página 478
- "Datos LOB en aplicaciones JDBC con el Controlador JDBC de DB2 Universal" en la página 315

Tareas relacionadas:

- "Conexión a una fuente de datos utilizando la interfaz DataSource" en la página 297
- "Conexión a una fuente de datos utilizando la interfaz DriverManager con el Controlador JDBC de DB2 Universal" en la página 294

Información relacionada:

- "Diferencias de JDBC entre el Controlador JDBC de DB2 Universal y otros controladores JDBC de DB2" en la página 459
- "Resumen de las extensiones del Controlador JDBC de DB2 Universal para JDBC" en la página 446

Comparación del soporte de controlador para las API de JDBC

Las tablas siguientes muestran las interfaces de JDBC y los controladores soportados para cada una. Los controladores y sus plataformas soportadas son:

Tabla 44. Controladores JDBC para DB2 UDB

Nombre del controlador JDBC	Plataforma DB2 UDB asociada
Controlador JDBC universal de DB2	DB2 UDB para Linux, UNIX y Windows o DB2 UDB para z/OS
Controlador JDBC/SQLJ 2.0 para OS/390	DB2 UDB para z/OS
Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows (uso desaconsejado)	DB2 UDB para Linux, UNIX y Windows

Tabla 45. Soporte de DB2 JDBC para métodos Array

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
getArray	No	No	No
getBaseType	No	No	No
getBaseTypeName	No	No	No
getResultSet	No	No	No

Tabla 46. Soporte de DB2 JDBC para métodos BatchUpdateException

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
Métodos heredados de java.lang.Exception	Sí	Sí	Sí
getUpdateCounts	Sí	Sí	Sí

Tabla 47. Soporte de DB2 JDBC para métodos Blob

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
getBinaryStream	Sí	Sí	Sí
getBytes	Sí	Sí	Sí
longitud	Sí	Sí	Sí
posición	Sí	Sí	Sí
setBinaryStream[1]	Sí	No	No
setBytes[1]	Sí	No	No
truncate[1]	Sí	No	No

Notas:

1. Esto es un método de JDBC 3.0.

Tabla 48. Soporte de DB2 JDBC para métodos CallableStatement

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
Métodos heredados de java.sql.Statement	Sí	Sí	Sí
Métodos heredados de java.sql.PreparedStatement	Sí	Sí	Sí
getArray	No	No	No
getBigDecimal	Sí	Sí	Sí
getBlob	Sí	Sí	Sí
getBoolean	Sí	Sí	Sí
getByte	Sí	Sí	Sí
getBytes	Sí	Sí	Sí
getClob	Sí	Sí	Sí
getDate	Sí	Sí	Sí
getDouble	Sí	Sí	Sí
getFloat	Sí	Sí	Sí
getInt	Sí	Sí	Sí
getLong	Sí	Sí	Sí
getObject	Sí[1]	Sí[1]	Sí[1]

Tabla 48. Soporte de DB2 JDBC para métodos CallableStatement (continuación)

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
getRef	No	No	No
getShort	Sí	Sí	Sí
getString	Sí	Sí	Sí
getTime	Sí	Sí	Sí
getTimestamp	Sí	Sí	Sí
registerOutParameter[2]	Sí	Sí	Sí
wasNull	Sí	Sí	Sí

Notas:

1. La siguiente modalidad del método `getObject` *no* está soportada:
   `getObject(int parameterIndex, java.util.Map map)`
2. La siguiente modalidad del método `registerOutParameter` *no* está soportada:
   `registerOutParameter(int parameterIndex, int jdbcType, String typeName)`

Tabla 49. Soporte de DB2 JDBC para métodos Clob

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
getAsciiStream	Sí	Sí	Sí
getCharacterStream	Sí	Sí	Sí
getSubString	Sí	Sí	Sí
longitud	Sí	Sí	Sí
posición	Sí	Sí	Sí
setAsciiStream[1]	Sí	No	No
setCharacterStream[1]	Sí	No	No
setString[1]	Sí	No	No
truncate[1]	Sí	No	No

Notas:

1. Esto es un método de JDBC 3.0.

Tabla 50. Soporte de DB2 JDBC para métodos Connection

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
clearWarnings	Sí	Sí	Sí
close	Sí	Sí	Sí
commit	Sí	Sí	Sí
createStatement	Sí[1]	Sí[2]	Sí

Tabla 50. Soporte de DB2 JDBC para métodos Connection (continuación)

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
getAutoCommit	Sí	Sí	Sí
getCatalog	Sí	Sí	Sí
getMetaData	Sí	Sí	Sí
getTransactionIsolation	Sí	Sí	Sí
getTypeMap	No	No	No
getWarnings	Sí	Sí	Sí
isClosed	Sí	Sí	Sí
isReadOnly	Sí	Sí	Sí
nativeSQL	Sí	Sí	Sí
prepareCall	Sí	Sí[3]	Sí
prepareStatement	Sí[4]	Sí	Sí
releaseSavepoint	Sí[5]	No	No
rollback	Sí	Sí[6]	Sí[6]
setAutoCommit	Sí	Sí	Sí
setCatalog	Sí	Sí	Sí
setReadOnly	Sí[7]	Sí[7]	Sí
setSavepoint	Sí[5]	No	No
setTransactionIsolation	Sí	Sí	Sí
setTypeMap	No	No	No

Notas:

1. Además de las modalidades de la sentencia `createStatement` para JDBC 2.0, está soportada la modalidad siguiente de `createStatement` para JDBC 3.0:

   ```
   createStatement(int resultSetType,
    int resultSetConcurrency,
    int resultSetHoldability)
   ```

2. Para la siguiente modalidad de `createStatement`, están soportados el valor `TYPE_FORWARD_ONLY` de *resultSetType* y el valor `CONCUR_READ_ONLY` de *resultSetConcurrency*:

   ```
   createStatement(int resultSetType, int resultSetConcurrency)
   ```

3. La siguiente modalidad de `prepareCall` *no* está soportada:

   ```
   prepareCall(String sql, int resultSetType, int resultSetConcurrency)
   ```

4. Además de las otras modalidades de `prepareStatement`, el Controlador JDBC universal de DB2 soporta la siguiente modalidad para JDBC 3.0:

   ```
   prepareStatement(String sql, int autoGeneratedKeys)
   ```

5. Esto es un método de JDBC 3.0.
6. El método `rollback(Savepoint punto_de_rescate)` de JDBC 3.0 no está soportado.
7. El controlador no utiliza el valor. Para el Controlador JDBC universal de DB2, una conexión se puede definir como de solo lectura mediante la propiedad `readOnly` de un objeto `Connection` o `DataSource`.

Tabla 51. Soporte de DB2 JDBC para métodos ConnectionEvent

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
Métodos heredados de java.util.EventObject	Sí	Sí	Sí
getSQLException	Sí	Sí	Sí

Tabla 52. Soporte de DB2 JDBC para métodos ConnectionEventListener

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
connectionClosed	Sí	Sí	Sí
connectionErrorOccurred	Sí	Sí	Sí

Tabla 53. Soporte de DB2 JDBC para métodos ConnectionPoolDataSource

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
getLoginTimeout	Sí	Sí	Sí
getLogWriter	Sí	Sí	Sí
getPooledConnection	Sí	Sí	Sí
setLoginTimeout	Sí[1]	Sí	Sí
setLogWriter	Sí	Sí	Sí

Nota:

1. Este método no está soportado para Conectividad de tipo 2 universal en DB2 UDB para el entorno OS/390 o z/OS.

Tabla 54. Soporte de DB2 JDBC para métodos de DatabaseMetaData

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
allProceduresAreCallable	Sí	Sí	Sí
allTablesAreSelectable	Sí	Sí	Sí
dataDefinitionCausesTransactionCommit	Sí	Sí	Sí
dataDefinitionIgnoredInTransactions	Sí	Sí	Sí
deletesAreDetected	Sí	Sí	Sí
doesMaxRowSizeIncludeBlobs	Sí	Sí	Sí
getAttributes	Sí	No	No
getBestRowIdentifier	Sí	Sí	Sí
getCatalogs	Sí	Sí	Sí

Tabla 54. Soporte de DB2 JDBC para métodos de DatabaseMetaData (continuación)

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
getCatalogSeparator	Sí	Sí	Sí
getCatalogTerm	Sí	Sí	Sí
getColumnPrivileges	Sí	Sí	Sí
getColumns	Sí[1]	Sí	Sí
getConnection	Sí	Sí	Sí
getCrossReference	Sí	Sí	Sí
getDatabaseMajorVersion	Sí	No	No
getDatabaseMinorVersion	Sí	No	No
getDatabaseProductName	Sí	Sí	Sí
getDatabaseProductVersion	Sí	Sí	Sí
getDefaultTransactionIsolation	Sí	Sí	Sí
getDriverMajorVersion	Sí	Sí	Sí
getDriverMinorVersion	Sí	Sí	Sí
getDriverName	Sí	Sí	Sí
getDriverVersion	Sí	Sí	Sí
getExportedKeys	Sí	Sí	Sí
getExtraNameCharacters	Sí	Sí	Sí
getIdentifierQuoteString	Sí	Sí	Sí
getImportedKeys	Sí	Sí	Sí
getIndexInfo	Sí	Sí	Sí
getJDBCMajorVersion	Sí	No	No
getJDBCMinorVersion	Sí	No	No
getMaxBinaryLiteralLength	Sí	Sí	Sí
getMaxCatalogNameLength	Sí	Sí	Sí
getMaxCharLiteralLength	Sí	Sí	Sí
getMaxColumnNameLength	Sí	Sí	Sí
getMaxColumnsInGroupBy	Sí	Sí	Sí
getMaxColumnsInIndex	Sí	Sí	Sí
getMaxColumnsInOrderBy	Sí	Sí	Sí
getMaxColumnsInSelect	Sí	Sí	Sí
getMaxColumnsInTable	Sí	Sí	Sí
getMaxConnections	Sí	Sí	Sí
getMaxCursorNameLength	Sí	Sí	Sí
getMaxIndexLength	Sí	Sí	Sí
getMaxProcedureNameLength	Sí	Sí	Sí
getMaxRowSize	Sí	Sí	Sí

Tabla 54. Soporte de DB2 JDBC para métodos de DatabaseMetaData (continuación)

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
getMaxSchemaNameLength	Sí	Sí	Sí
getMaxStatementLength	Sí	Sí	Sí
getMaxStatements	Sí	Sí	Sí
getMaxTableNameLength	Sí	Sí	Sí
getMaxTablesInSelect	Sí	Sí	Sí
getMaxUserNameLength	Sí	Sí	Sí
getNumericFunctions	Sí	Sí	Sí
getPrimaryKeys	Sí	Sí	Sí
getProcedureColumns	Sí	Sí	Sí
getProcedures	Sí	Sí	Sí
getProcedureTerm	Sí	Sí	Sí
getResultSetHoldability	Sí	No	No
getSchemas	Sí[1]	Sí	Sí
getSchemaTerm	Sí	Sí	Sí
getSearchStringEscape	Sí	Sí	Sí
getSQLKeywords	Sí	Sí	Sí
getSQLStateType	Sí	No	No
getStringFunctions	Sí	Sí	Sí
getSuperTables	Sí[2]	No	No
getSuperTypes	Sí[2]	No	No
getSystemFunctions	Sí	Sí	Sí
getTablePrivileges	Sí	Sí	Sí
getTables	Sí[1]	Sí	Sí
getTableTypes	Sí	Sí	Sí
getTimeDateFunctions	Sí	Sí	Sí
getTypeInfo	Sí	Sí	Sí
getUDTs	No	No	Sí[2]
getURL	Sí	Sí	Sí
getUserName	Sí	Sí	Sí
getVersionColumns	Sí	Sí	Sí
insertsAreDetected	Sí	Sí	Sí
isCatalogAtStart	Sí	Sí	Sí
isReadOnly	Sí	Sí	Sí
nullPlusNonNullIsNull	Sí	Sí	Sí
nullsAreSortedAtEnd	Sí	Sí	Sí
nullsAreSortedAtStart	Sí	Sí	Sí

Tabla 54. Soporte de DB2 JDBC para métodos de DatabaseMetaData (continuación)

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
nullsAreSortedHigh	Sí	Sí	Sí
nullsAreSortedLow	Sí	Sí	Sí
othersDeletesAreVisible	Sí	Sí	Sí
othersInsertsAreVisible	Sí	Sí	Sí
othersUpdatesAreVisible	Sí	Sí	Sí
ownDeletesAreVisible	Sí	Sí	Sí
ownInsertsAreVisible	Sí	Sí	Sí
ownUpdatesAreVisible	Sí	Sí	Sí
storesLowerCaseIdentifiers	Sí	Sí	Sí
storesLowerCaseQuotedIdentifiers	Sí	Sí	Sí
storesMixedCaseIdentifiers	Sí	Sí	Sí
storesMixedCaseQuotedIdentifiers	Sí	Sí	Sí
storesUpperCaseIdentifiers	Sí	Sí	Sí
storesUpperCaseQuotedIdentifiers	Sí	Sí	Sí
supportsAlterTableWithAddColumn	Sí	Sí	Sí
supportsAlterTableWithDropColumn	Sí	Sí	Sí
supportsANSI92EntryLevelSQL	Sí	Sí	Sí
supportsANSI92FullSQL	Sí	Sí	Sí
supportsANSI92IntermediateSQL	Sí	Sí	Sí
supportsBatchUpdates	Sí	Sí	Sí
supportsCatalogsInDataManipulation	Sí	Sí	Sí
supportsCatalogsInIndexDefinitions	Sí	Sí	Sí
supportsCatalogsInPrivilegeDefinitions	Sí	Sí	Sí
supportsCatalogsInProcedureCalls	Sí	Sí	Sí
supportsCatalogsInTableDefinitions	Sí	Sí	Sí
SupportsColumnAliasing	Sí	Sí	Sí
supportsConvert	Sí	Sí	Sí
supportsCoreSQLGrammar	Sí	Sí	Sí
supportsCorrelatedSubqueries	Sí	Sí	Sí
supportsDataDefinitionAndDataManipulationTransactions	Sí	Sí	Sí
supportsDataManipulationTransactionsOnly	Sí	Sí	Sí
supportsDifferentTableCorrelationNames	Sí	Sí	Sí
supportsExpressionsInOrderBy	Sí	Sí	Sí
supportsExtendedSQLGrammar	Sí	Sí	Sí
supportsFullOuterJoins	Sí	Sí	Sí
supportsGetGeneratedKeys	Sí	No	No

Tabla 54. Soporte de DB2 JDBC para métodos de DatabaseMetaData (continuación)

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
supportsGroupBy	Sí	Sí	Sí
supportsGroupByBeyondSelect	Sí	Sí	Sí
supportsGroupByUnrelated	Sí	Sí	Sí
supportsIntegrityEnhancementFacility	Sí	Sí	Sí
supportsLikeEscapeClause	Sí	Sí	Sí
supportsLimitedOuterJoins	Sí	Sí	Sí
supportsMinimumSQLGrammar	Sí	Sí	Sí
supportsMixedCaseIdentifiers	Sí	Sí	Sí
supportsMixedCaseQuotedIdentifiers	Sí	Sí	Sí
supportsMultipleOpenResults	Sí	Sí	No
supportsMultipleResultSets	Sí	Sí	Sí
supportsMultipleTransactions	Sí	Sí	Sí
supportsNamedParameters	Sí	No	No
supportsNonNullableColumns	Sí	Sí	Sí
supportsOpenCursorsAcross Commit	Sí	Sí	Sí
supportsOpenCursorsAcross Rollback	Sí	Sí	Sí
supportsOpenStatementsAcrossCommit	Sí	Sí	Sí
supportsOpenStatementsAcrossRollback	Sí	Sí	Sí
supportsOrderByUnrelated	Sí	Sí	Sí
supportsOuterJoins	Sí	Sí	Sí
supportsPositionedDelete	Sí	Sí	Sí
supportsPositionedUpdate	Sí	Sí	Sí
supportsResultSetConcurrency	Sí	Sí	Sí
supportsResultSetHoldability	Sí	No	No
supportsResultSetType	Sí	Sí	Sí
supportsSavepoints	Sí	No	No
supportsSchemasInDataManipulation	Sí	Sí	Sí
supportsSchemasInIndexDefinitions	Sí	Sí	Sí
supportsSchemasInPrivilegeDefinitions	Sí	Sí	Sí
supportsSchemasInProcedureCalls	Sí	Sí	Sí
supportsSchemasInTableDefinitions	Sí	Sí	Sí
supportsSelectForUpdate	Sí	Sí	Sí
supportsStoredProcedures	Sí	Sí	Sí
supportsSubqueriesInComparisons	Sí	Sí	Sí
supportsSubqueriesInExists	Sí	Sí	Sí
supportsSubqueriesInIns	Sí	Sí	Sí

Tabla 54. Soporte de DB2 JDBC para métodos de DatabaseMetaData (continuación)

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
supportsSubqueriesInQuantifieds	Sí	Sí	Sí
supportsSuperTables	Sí	No	No
supportsSuperTypes	Sí	No	No
supportsTableCorrelationNames	Sí	Sí	Sí
supportsTransactionIsolationLevel	Sí	Sí	Sí
supportsTransactions	Sí	Sí	Sí
supportsUnion	Sí	Sí	Sí
supportsUnionAll	Sí	Sí	Sí
updatesAreDetected	Sí	Sí	Sí
usesLocalFilePerTable	Sí	Sí	Sí
usesLocalFiles	Sí	Sí	Sí

Notas:

1. La versión de este método para JDBC 3.0 está soportada.
2. El método se puede ejecutar, pero devuelve un `conjunto de resultados` vacío.

Tabla 55. Soporte de DB2 JDBC para métodos DataSource

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
getConnection	Sí	Sí	Sí
getLoginTimeout	Sí	Sí	Sí[1]
getLogWriter	Sí	Sí	Sí
setLoginTimeout	Sí[2]	Sí	Sí[1]
setLogWriter	Sí	Sí	Sí

Notas:

1. El Controlador JDBC de DB2 de tipo 2 no utiliza este valor.
2. Este método no está soportado para Conectividad de tipo 2 universal en DB2 UDB para el entorno OS/390 o z/OS.

Tabla 56. Soporte de DB2 JDBC para métodos DataTruncation

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
Métodos heredados de java.lang.Throwable	Sí	Sí	Sí
Métodos heredados de java.sql.SQLException	Sí	Sí	Sí

Tabla 56. Soporte de DB2 JDBC para métodos DataTruncation (continuación)

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
Métodos heredados de java.sql.SQLWarning	Sí	Sí	Sí
getDataSize	Sí	Sí	Sí
getIndex	Sí	Sí	Sí
getParameter	Sí	Sí	Sí
getRead	Sí	Sí	Sí
getTransferSize	Sí	Sí	Sí

Tabla 57. Soporte de DB2 JDBC para métodos Driver

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
acceptsURL	Sí	Sí	Sí
connect	Sí	Sí	Sí
getMajorVersion	Sí	Sí	Sí
getMinorVersion	Sí	Sí	Sí
getPropertyInfo	Sí	Sí	Sí
jdbcCompliant	Sí	Sí	Sí

Tabla 58. Soporte de DB2 JDBC para métodos DriverManager

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
deregisterDriver	Sí	Sí	Sí
getConnection	Sí	Sí	Sí
getDriver	Sí	Sí	Sí
getDrivers	Sí	Sí	Sí
getLoginTimeout	Sí	Sí	Sí[1]
getLogStream	Sí	Sí	Sí
getLogWriter	Sí	Sí	Sí
println	Sí	Sí	Sí
registerDriver	Sí	Sí	Sí
setLoginTimeout	Sí[2]	Sí	Sí[1]
setLogStream	Sí	Sí	Sí
setLogWriter	Sí	Sí	Sí

Notas:

1. El Controlador JDBC de DB2 de tipo 2 no utiliza este valor.
2. Este método no está soportado para Conectividad de tipo 2 universal en DB2 UDB para el entorno OS/390 o z/OS.

Tabla 59. Soporte de DB2 JDBC para métodos de ParameterMetaData

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
getParameterClassName	No	No	No
getParameterCount	Sí	No	No
getParameterMode	Sí	No	No
getParameterType	Sí	No	No
getParameterTypeName	Sí	No	No
getPrecision	Sí	No	No
getScale	Sí	No	No
isNullable	Sí	No	No
isSigned	Sí	No	No

Tabla 60. Soporte de DB2 JDBC para métodos PooledConnection

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
addConnectionEventListener	Sí	Sí	Sí
close	Sí	Sí	Sí
getConnection	Sí	Sí	Sí
removeConnectionEventListener	Sí	Sí	Sí

Tabla 61. Soporte de DB2 JDBC para métodos PreparedStatement

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
Métodos heredados de java.sql.Statement	Sí	Sí	Sí
addBatch	Sí	Sí	Sí
clearParameters	Sí	Sí	Sí
execute	Sí	Sí	Sí
executeQuery	Sí	Sí	Sí
executeUpdate	Sí	Sí	Sí
getMetaData	Sí	Sí	Sí
setArray	No	No	No
setAsciiStream	Sí	Sí	Sí
setBigDecimal	Sí	Sí	Sí
setBinaryStream	Sí	Sí	Sí
setBlob	Sí	Sí	Sí
setBoolean	Sí	Sí	Sí

Tabla 61. Soporte de DB2 JDBC para métodos PreparedStatement (continuación)

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
setByte	Sí	Sí	Sí
setBytes	Sí	Sí	Sí
setCharacterStream	Sí	Sí	Sí
setClob	Sí	Sí	Sí
setDate	Sí	Sí[1]	Sí
setDouble	Sí	Sí	Sí
setFloat	Sí	Sí	Sí
setInt	Sí	Sí	Sí
setLong	Sí	Sí	Sí
setNull	Sí[2]	Sí[2]	Sí[2]
setObject	Sí	Sí	Sí
setRef	No	No	No
setShort	Sí	Sí	Sí
setString	Sí[3]	Sí[3]	Sí[3]
setTime	Sí[4]	Sí[4]	Sí
setTimestamp	Sí[5]	Sí[5]	Sí
setUnicodeStream	Sí	Sí	Sí
setURL	Sí	No	Sí

Notas:

1. La siguiente modalidad de `setDate` *no* está soportada:
   `setDate(int parameterIndex, java.sql.Date x, java.util.Calendar cal)`
2. La siguiente modalidad de `setNull` *no* está soportada:
   `setNull(int parameterIndex, int jdbcType, String typeName)`
3. setString no está soportado si la columna tiene el atributo FOR BIT DATA o el tipo de datos es BLOB.
4. La siguiente modalidad de `setTime` *no* está soportada:
   `setTime(int parameterIndex, java.sql.Time x, java.util.Calendar cal)`
5. La siguiente modalidad de `setTimestamp` *no* está soportada:
   `setTimestamp(int parameterIndex, java.sql.Timestamp x, java.util.Calendar cal)`

Tabla 62. Soporte de DB2 JDBC para métodos Ref

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
get BaseTypeName	No	No	No

Tabla 63. Soporte de DB2 JDBC para métodos ResultSet

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
absolute	Sí	No	Sí
afterLast	Sí	No	Sí
beforeFirst	Sí	No	Sí
cancelRowUpdates	Sí	No	No
clearWarnings	Sí	Sí	Sí
close	Sí	Sí	Sí
deleteRow	Sí	No	No
findColumn	Sí	Sí	Sí
first	Sí	No	Sí
getArray	No	No	No
getAsciiStream	Sí	Sí	Sí
getBigDecimal	Sí	Sí	Sí
getBinaryStream	Sí[1]	Sí[1]	Sí
getBlob	Sí	Sí	Sí
getBoolean	Sí	Sí	Sí
getByte	Sí	Sí	Sí
getBytes	Sí	Sí	Sí
getCharacterStream	Sí	Sí	Sí
getClob	Sí	Sí	Sí
getConcurrency	Sí	Sí	Sí
getCursorName	Sí	Sí	Sí
getDate	Sí	Sí[2]	Sí
getDouble	Sí	Sí	Sí
getFetchDirection	Sí	Sí	Sí
getFetchSize	Sí	Sí	Sí
getFloat	Sí	Sí	Sí
getInt	Sí	Sí	Sí
getLong	Sí	Sí	Sí
getMetaData	Sí	Sí	Sí
getObject	Sí[3]	Sí[3]	Sí[3]
getRef	No	No	No
getRow	Sí	No	Sí
getShort	Sí	Sí	Sí
getStatement	Sí	Sí	Sí
getString	Sí	Sí	Sí
getTime	Sí	Sí[4]	Sí
getTimestamp	Sí	Sí[5]	Sí
getType	Sí	Sí	Sí

Tabla 63. Soporte de DB2 JDBC para métodos ResultSet (continuación)

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
getUnicodeStream	Sí	Sí	Sí
getURL	Sí	No	Sí
getWarnings	Sí	Sí	Sí
insertRow	No	No	No
isAfterLast	Sí	No	Sí
isBeforeFirst	Sí	No	Sí
isFirst	Sí	No	Sí
isLast	Sí	No	Sí
last	Sí	No	Sí
moveToCurrentRow	Sí	No	No
moveToInsertRow	No	No	No
next	Sí	Sí	Sí
previous	Sí	No	Sí
refreshRow	Sí	No	No
relative	Sí	No	Sí
rowDeleted	Sí	No	No
rowInserted	No	No	No
rowUpdated	Sí	No	No
setFetchDirection	Sí	Sí[6]	Sí
setFetchSize	Sí	Sí	Sí
updateAsciiStream	Sí	No	No
updateBigDecimal	Sí	No	No
updateBinaryStream	Sí	No	No
updateBoolean	Sí	No	No
updateByte	Sí	No	No
updateBytes	Sí	No	No
updateCharacterStream	Sí	No	No
updateDate	Sí	No	No
updateDouble	Sí	No	No
updateFloat	Sí	No	No
updateInt	Sí	No	No
updateLong	Sí	No	No
updateNull	Sí	No	No
updateObject	Sí	No	No
updateRow	Sí	No	No
updateShort	Sí	No	No
updateString	Sí	No	No
updateTime	Sí	No	No

Tabla 63. Soporte de DB2 JDBC para métodos ResultSet (continuación)

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
updateTimestamp	Sí	No	No
wasNull	Sí	Sí	Sí

Notas:

1. `getBinaryStream` no está soportado para columnas de tipo CLOB.
2. Las siguientes modalidades de `getDate` *no* están soportadas:

   ```
   getDate(int columnIndex, java.util.Calendar cal)
   getDate(String columnName, java.util.Calendar cal)
   ```

3. La siguiente modalidad del método `getObject` *no* está soportada:

   ```
   getObject(int parameterIndex, java.util.Map  map)
   ```

4. Las siguientes modalidades de `getTime` *no* están soportadas:

   ```
   getTime(int columnIndex, java.util.Calendar cal)
   getTime(String columnName, java.util.Calendar cal)
   ```

5. Las siguientes modalidades de `getTimestamp` *no* están soportadas:

   ```
   getTimestamp(int columnIndex, java.util.Calendar cal)
   getTimestamp(String columnName, java.util.Calendar cal)
   ```

6. Soportado solamente si *direction* es `ResultSet.FETCH_FORWARD`.

Tabla 64. Soporte de DB2 JDBC para métodos de ResultSetMetaData

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
getCatalogName	Sí	Sí	Sí
getColumnClassName	No	No	Sí
getColumnCount	Sí	Sí	Sí
getColumnDisplaySize	Sí	Sí	Sí
getColumnLabel	Sí	Sí	Sí
getColumnName	Sí	Sí	Sí
getColumnType	Sí	Sí	Sí
getColumnTypeName	Sí	Sí	Sí
getPrecision	Sí	Sí	Sí
getScale	Sí	Sí	Sí
getSchemaName	Sí	Sí	Sí
getTableName	Sí	Sí	Sí
isAutoIncrement	Sí	Sí	Sí
isCaseSensitive	Sí	Sí	Sí
isCurrency	Sí	Sí	Sí
isDefinitelyWritable	Sí	Sí	Sí
isNullable	Sí	Sí	Sí
isReadOnly	Sí	Sí	Sí
isSearchable	Sí	Sí	Sí

Tabla 64. Soporte de DB2 JDBC para métodos de ResultSetMetaData (continuación)

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
isSigned	Sí	Sí	Sí
isWritable	Sí	Sí	Sí

Tabla 65. Soporte de DB2 JDBC para métodos SQLData

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
getSQLTypeName	No	No	No
readSQL	No	No	No
writeSQL	No	No	No

Tabla 66. Soporte de DB2 JDBC para métodos SQLException

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
Métodos heredados de java.lang.Exception	Sí	Sí	Sí
getSQLState	Sí	Sí	Sí
getErrorCode	Sí	Sí	Sí
getNextException	Sí	Sí	Sí
setNextException	Sí	Sí	Sí

Tabla 67. Soporte de DB2 JDBC para métodos SQLInput

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
readArray	No	No	No
readAsciiStream	No	No	No
readBigDecimal	No	No	No
readBinaryStream	No	No	No
readBlob	No	No	No
readBoolean	No	No	No
readByte	No	No	No
readBytes	No	No	No
readCharacterStream	No	No	No
readClob	No	No	No
readDate	No	No	No
readDouble	No	No	No

Tabla 67. Soporte de DB2 JDBC para métodos SQLInput (continuación)

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
readFloat	No	No	No
readInt	No	No	No
readLong	No	No	No
readObject	No	No	No
readRef	No	No	No
readShort	No	No	No
readString	No	No	No
readTime	No	No	No
readTimestamp	No	No	No
wasNull	No	No	No

Tabla 68. Soporte de DB2 JDBC para métodos SQLOutput

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
writeArray	No	No	No
writeAsciiStream	No	No	No
writeBigDecimal	No	No	No
writeBinaryStream	No	No	No
writeBlob	No	No	No
writeBoolean	No	No	No
writeByte	No	No	No
writeBytes	No	No	No
writeCharacterStream	No	No	No
writeClob	No	No	No
writeDate	No	No	No
writeDouble	No	No	No
writeFloat	No	No	No
writeInt	No	No	No
writeLong	No	No	No
writeObject	No	No	No
writeRef	No	No	No
writeShort	No	No	No
writeString	No	No	No
writeStruct	No	No	No
writeTime	No	No	No
writeTimestamp	No	No	No

Tabla 69. Soporte de DB2 JDBC para métodos Statement

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
addBatch	Sí	Sí	Sí
cancel	Sí[1]	No	Sí
clearBatch	Sí	Sí	Sí
clearWarnings	Sí	Sí	Sí
close	Sí	Sí	Sí
execute	Sí[2]	Sí	Sí
executeBatch	Sí	Sí	Sí
executeQuery	Sí	Sí	Sí
executeUpdate	Sí[2]	Sí	Sí
getConnection	Sí	No	Sí
getFetchDirection	Sí	No	Sí
getFetchSize	Sí	No	Sí
getGeneratedKeys	Sí	No	No
getMaxFieldSize	Sí	Sí	Sí
getMaxRows	Sí	Sí	Sí
getMoreResults	Sí[3]	Sí	Sí
getQueryTimeout	Sí[1]	Sí	Sí
getResultSet	Sí	Sí	Sí
getResultSetConcurrency	Sí	Sí	Sí
getResultSetType	Sí	Sí	Sí
getUpdateCount[4]	Sí	Sí	Sí
getWarnings	Sí	Sí	Sí
setCursorName	Sí	Sí	Sí
setEscapeProcessing	Sí	Sí	Sí
setFetchDirection	Sí	Sí	Sí
setFetchSize	Sí	No	Sí
setMaxFieldSize	Sí	Sí	Sí
setMaxRows	Sí	Sí	Sí
setQueryTimeout	Sí[5]	Sí[5]	Sí

Tabla 69. Soporte de DB2 JDBC para métodos Statement (continuación)

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows

Notas:

1. Este método no está soportado para Conectividad de tipo 2 universal en el entorno OS/390 o z/OS.
2. Además de las otras modalidades de `execute` o `executeUpdate`, el Controlador JDBC universal de DB2 soporta las siguientes modalidades para JDBC 3.0:

   ```
   executeUpdate(String sql, int autoGeneratedKeys)
   execute(String sql, int autoGeneratedKeys)
   ```
3. Además de `getMoreResults()`, el Controlador JDBC universal de DB2 soporta las siguientes modalidades para JDBC 3.0:
   - `getMoreResults(java.sql.Statement.CLOSE_CURRENT_RESULT)`
   - `getMoreResults(java.sql.Statement.KEEP_CURRENT_RESULT)`
   - `getMoreResults(java.sql.Statement.CLOSE_ALL_RESULTS)`
4. No está soportado para `conjuntos de resultados` de procedimiento almacenado.
5. Soportado solamente para un valor de *seconds* igual a 0.

Tabla 70. Soporte de DB2 JDBC para métodos Struct

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
getSQLTypeName	No	No	No
getAttributes	No	No	No

Tabla 71. Soporte de DB2 JDBC para métodos XAConnection

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
Métodos heredados de javax.sql.PooledConnection	Sí[1]	No	Sí
getXAResource	Sí[1]	No	Sí

Notas:

1. Este método está soportado para la Conectividad de tipo 2 del controlador JDBC universal de DB2 con un servidor DB2 UDB para Linux, UNIX y Windows, o para la Conectividad de tipo 4 del controlador JDBC universal de DB2 con un servidor DB2 UDB para z/OS.

Tabla 72. Soporte de DB2 JDBC para métodos XADataSource

Método JDBC	Soporte de Controlador JDBC universal de DB2	Soporte de Controlador JDBC/SQLJ 2.0 para OS/390	Soporte de Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows
getLoginTimeout	Sí	No	Sí
getLogWriter	Sí	No	Sí
getXAConnection	Sí	No	Sí
setLoginTimeout	Sí	No	Sí
setLogWriter	Sí	No	Sí

Información relacionada:

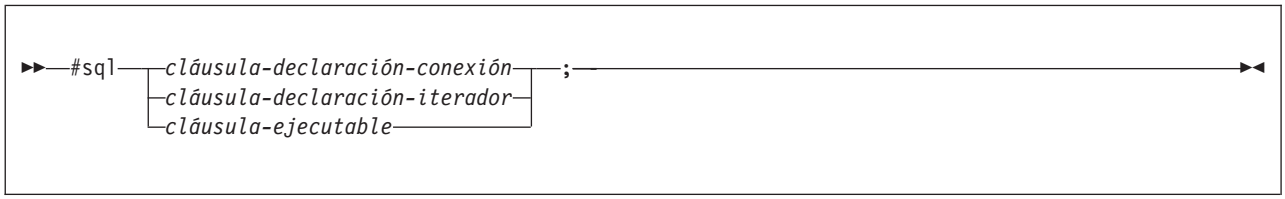
- "Diferencias de JDBC entre el Controlador JDBC de DB2 Universal y otros controladores JDBC de DB2" en la página 459

Información de consulta sobre sentencias de SQLJ

Las secciones siguientes contienen información sobre la sintaxis de cláusulas de SQLJ.

Cláusula SQLJ

Las sentencias de SQL de un programa SQLJ están contenidas en cláusulas SQLJ. La sintaxis general de una cláusula SQLJ es:



Las palabras clave de una cláusula SQLJ distinguen entre mayúsculas y minúsculas, a menos que esas palabras clave formen parte de una sentencia de SQL dentro de una cláusula ejecutable.

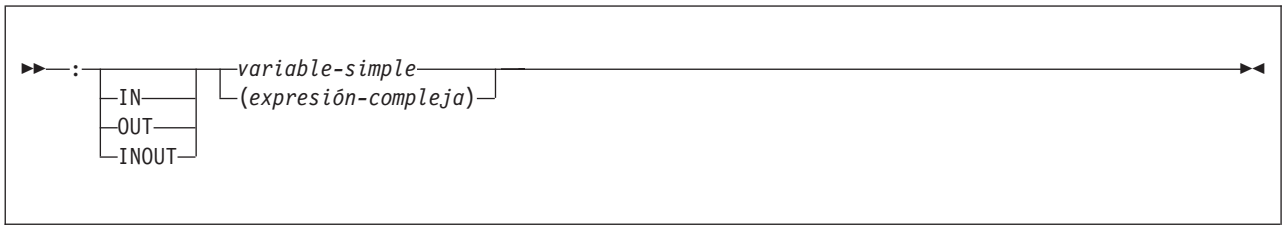
Información relacionada:

- "Cláusula connection-declaration de SQLJ" en la página 430
- "Cláusula ejecutable de SQLJ" en la página 433
- "Cláusula de declaración de iterador de SQLJ" en la página 431

Expresión de lenguaje principal de SQLJ

Una expresión de lenguaje principal es una variable o expresión Java que se especifica en cláusulas de SQLJ dentro de un programa de aplicación SQLJ.

Sintaxis:



Descripción:

: Indica que la variable o expresión que sigue a continuación es una expresión de lenguaje principal. La variable o expresión debe estar precedida inmediatamente por los dos puntos (:).

IN | OUT | INOUT
Para expresiones de lenguaje principal que se utilizan como parámetros en una llamada de procedimiento almacenado, identifica si el parámetro proporciona

datos al procedimiento almacenado (IN), recibe datos del procedimiento almacenado (OUT), o realiza ambas cosas (INOUT). El valor por omisión es IN.

variable-simple
Especifica un identificador Java no calificado.

expresión-compleja
Especifica una expresión Java cuya evaluación da como resultado un valor individual.

Notas de uso:

- Las expresiones complejas deben estar encerradas entre paréntesis.
- La ubicación de una expresión de lenguaje principal dentro de una sentencia de SQL estático está regida por reglas de ANSI/ISO.

Conceptos relacionados:

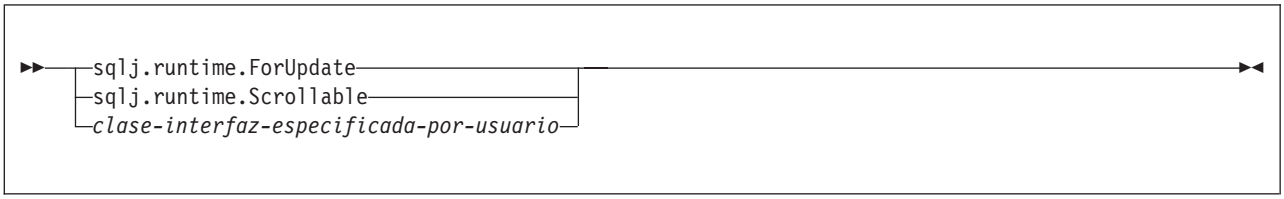
- "Variables en aplicaciones SQLJ" en la página 347

Cláusula implements de SQLJ

La cláusula implements obtiene una o más clases a partir de una interfaz Java.

Sintaxis:

elemento-interfaz:



Descripción:

elemento-interfaz
Especifica una interfaz Java definida por el usuario, la interfaz `sqlj.runtime.ForUpdate` de SQLJ o la interfaz `sqlj.runtime.Scrollable` de SQLJ.

Es necesario que implemente `sqlj.runtime.ForUpdate` cuando declare un iterador para una operación UPDATE o DELETE de posición. Consulte el tema Ejecución de operaciones UPDATE y DELETE de posición en una aplicación SQLJ para conocer cómo ejecutar una operación UPDATE o DELETE de posición en SQLJ.

Es necesario que implemente `sqlj.runtime.Scrollable` cuando declare un iterador desplazable. Consulte el tema Utilización de iteradores desplazables en una aplicación SQLJ para obtener información sobre iteradores desplazables.

Tareas relacionadas:

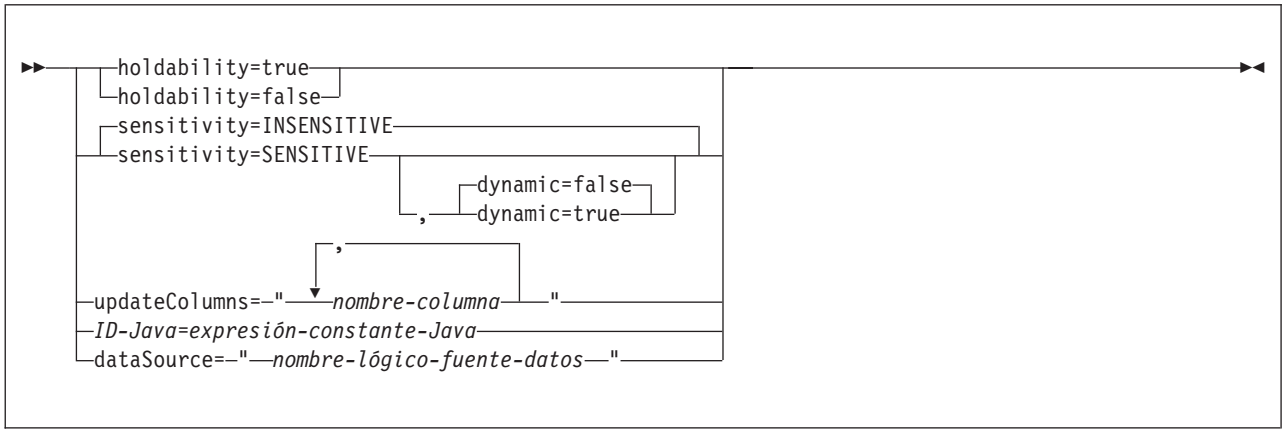
- "Ejecución de operaciones UPDATE y DELETE de posición en una aplicación SQLJ" en la página 363
- "Utilización de iteradores desplazables en una aplicación SQLJ" en la página 389

Cláusula with de SQLJ

La cláusula with especifica uno o más atributos para un iterador o contexto de conexión.

Sintaxis:

elemento-with:



Descripción:

holdability
Para un iterador, especifica si el iterador conserva su posición en una tabla después de ejecutarse una operación COMMIT. El valor de holdability debe ser `true` o `false`.

sensitivity
Para un iterador, especifica si los cambios hechos en la tabla subyacente pueden ser visibles para el iterador después de abrir el iterador. El valor debe ser INSENSITIVE o SENSITIVE. El valor por omisión es INSENSITIVE.

dynamic
Para un iterador que esté definido con sensitivity=SENSITIVE, este atributo especifica si se cumplen las condiciones siguientes:

- Cuando la aplicación ejecuta sentencias UPDATE y DELETE de posición con el iterador, esos cambios son visibles para el iterador.
- Cuando la aplicación ejecuta sentencias INSERT, UPDATE y DELETE dentro de la aplicación, pero fuera del iterador, esos cambios son visibles para el iterador.

El valor de dynamic debe ser `true` o `false`. El valor por omisión es `false`.

Si el valor de dynamic es `true`, la fuente de datos debe dar soporte a cursores dinámicos desplazables.

updateColumns
Para un iterador, especifica las columnas que se deben modificar cuando el iterador se utiliza para una sentencia UPDATE de posición. El valor de updateColumns debe ser una cadena de caracteres literal que contenga los nombres de las columnas, separados por comas.

nombre-columna
Para un iterador, especifica una columna de la tabla de resultados que se debe actualizar utilizando el iterador.

ID-Java
Para un iterador o contexto de conexión, especifica una variable Java que identifica un atributo, definido por el usuario, del iterador o contexto de conexión. El valor de *expresión-constante-Java* también está definido por el usuario.

dataSource
Para un contexto de conexión, especifica el nombre lógico de un objeto `DataSource`, creado por separado, que representa la fuente de datos a la que se conectará la aplicación. Esta opción solo está disponible para el Controlador JDBC universal de DB2.

Notas de uso:

- El valor situado en el lado izquierdo de un elemento with debe ser exclusivo dentro de su cláusula.
- Si especifica updateColumns en un elemento with de una cláusula de declaración de iterador, esta cláusula también debe contener una cláusula implements en la que se especifique la interfaz sqlj.runtime.ForUpdate.
- Si el usuario no personaliza su programa SQLJ, el controlador JDBC no tiene en cuenta el valor de holdability que está contenido en la cláusula with. En lugar de ello, el controlador utiliza el valor de holdability definido para el controlador JDBC.

Conceptos relacionados:

- "Uso de SQLJ y JDBC en la misma aplicación" en la página 373

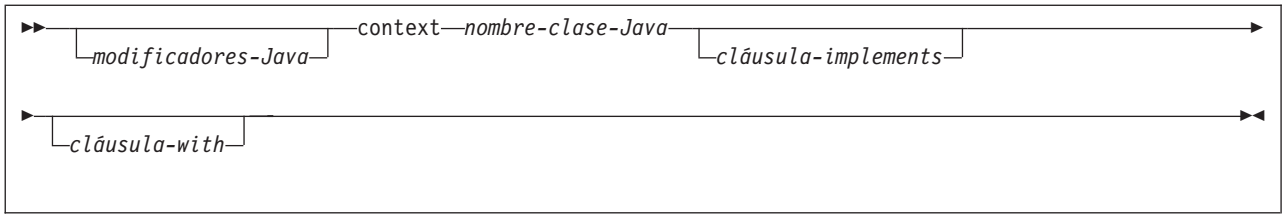
Tareas relacionadas:

- "Conexión a una fuente de datos utilizando SQLJ" en la página 349
- "Ejecución de operaciones UPDATE y DELETE de posición en una aplicación SQLJ" en la página 363
- "Utilización de iteradores desplazables en una aplicación SQLJ" en la página 389

Cláusula connection-declaration de SQLJ

La cláusula connection-declaration declara una conexión con una fuente de datos en un programa de aplicación de SQLJ.

Sintaxis:



Descripción:

modificadores-Java
Especifica modificadores que son válidos para declaraciones de clases Java, tales como static, public, private o protected.

nombre-clase-Java
Especifica un identificador Java válido. Durante el proceso de preparación del programa, SQLJ genera una clase de contexto de conexión cuyo nombre es este identificador.

cláusula-implements
Consulte el tema Cláusula implements de SQLJ para obtener una descripción de esta cláusula. En una cláusula de declaración de conexión, la clase de interfaz referida por la cláusula implements debe ser una clase de interfaz definida por el usuario.

cláusula-with
Consulte el tema Cláusula with de SQLJ para obtener una descripción de esta cláusula.

Notas de uso:

- SQLJ genera una declaración de clase de conexión para cada cláusula de declaración de conexión especificada por el usuario. Las conexiones con una fuente de datos de SQLJ son objetos de esas clases de conexión generadas.
- Puede especificar una cláusula de declaración de conexión en cualquier lugar de un programa Java donde pueda existir una definición de clase Java.

Tareas relacionadas:

- "Conexión a una fuente de datos utilizando SQLJ" en la página 349

Información relacionada:

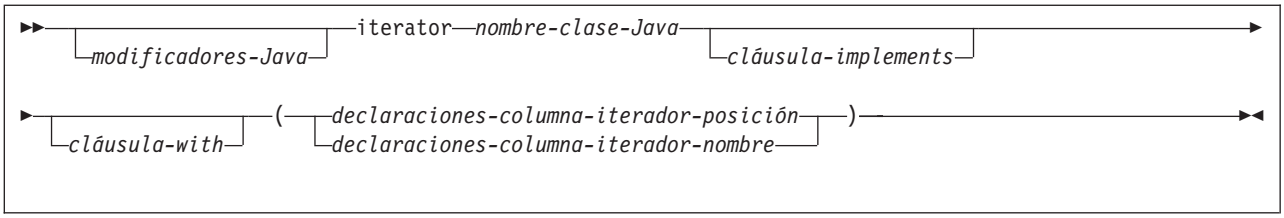
- "Cláusula implements de SQLJ" en la página 428
- "Cláusula with de SQLJ" en la página 429

Cláusula de declaración de iterador de SQLJ

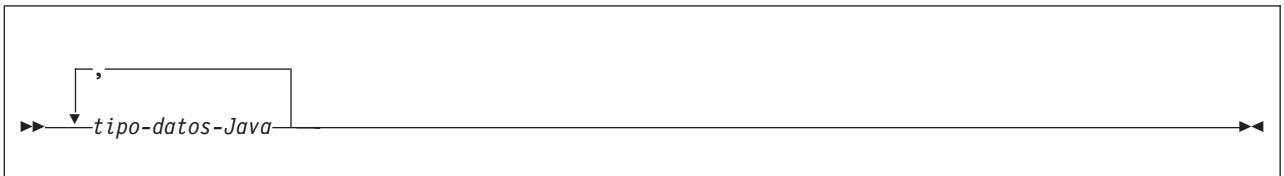
Una cláusula de declaración de iterador declara una clase de iterador de posición o de iterador de nombre dentro de un programa de aplicación SQLJ. Un iterador contiene la tabla de resultados de una consulta. SQLJ genera una clase de iterador para cada cláusula de declaración de iterador especificada por el usuario. Un iterador es un objeto de una clase de iterador.

Una cláusula de declaración de iterador tiene formatos diferentes para un iterador de posición y un iterador de nombre. Las dos clases de iteradores son tipos Java diferentes e incompatibles que se implementan con interfaces diferentes.

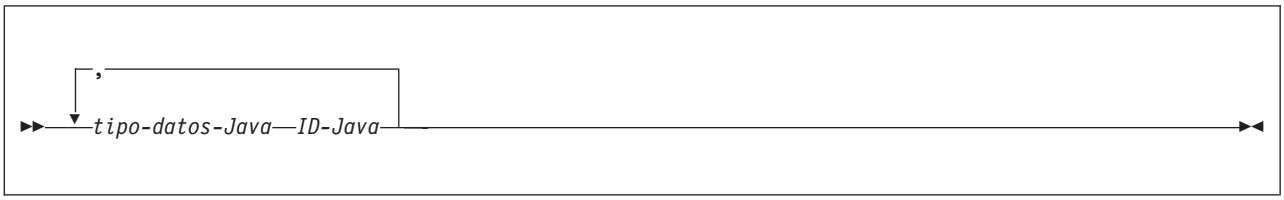
Sintaxis:



Declaraciones de columna-iterador-posición:



Declaraciones de columna-iterador-nombre:



Descripción:

modificadores-Java
Especifica modificadores cualesquiera que sean válidos para declaraciones de clases Java, tales como static, public, private o protected.

nombre-clase-Java
Especifica un identificador Java cualquiera que sea válido. Durante el proceso de preparación del programa, SQLJ genera una clase de iterador cuyo nombre es este identificador.

cláusula-implements
Consulte el tema Cláusula implements de SQLJ para obtener una descripción de esta cláusula. Para una cláusula de declaración de iterador que declara un iterador para una operación UPDATE o DELETE de posición, la cláusula implements debe especificar la interfaz `sqlj.runtime.ForUpdate`. Para una cláusula de declaración de iterador que declara un iterador desplazable, la cláusula implements debe especificar la interfaz `sqlj.runtime.Scrollable`.

cláusula-with
Consulte el tema Cláusula with de SQLJ para obtener una descripción de esta cláusula.

declaraciones-columna-iterador-posición
Especifica una lista de tipos de datos Java, que son los tipos de datos de las columnas del iterador de posición. Los tipos de datos contenidos en la lista deben estar separados por comas. El orden de los tipos de datos en la declaración de iterador de posición es el mismo que el orden de las columnas en la tabla de resultados. Para que sea efectiva la comprobación en línea durante la personalización del perfil serializado, los tipos de datos de las columnas del iterador deben ser compatibles con los tipos de datos de las columnas de la tabla de resultados. Consulte el tema Tipos de datos de Java, JDBC y SQL para obtener una lista de los tipos de datos compatibles.

declaraciones-columna-iterador-nombre
Especifica una lista de tipos de datos Java e identificadores Java, que son los tipos de datos y nombres de las columnas del iterador de nombre. Los pares tipo de datos-nombre deben estar separador por comas. El nombre de una columna del iterador debe coincidir con el nombre de una columna de la tabla de resultados, excepto en lo que respecta al uso de letras mayúsculas y minúsculas. Para que sea efectiva la comprobación en línea durante la personalización del perfil serializado, los tipos de datos de las columnas del iterador deben ser compatibles con los tipos de datos de las columnas de la tabla de resultados. Consulte el tema Tipos de datos de Java, JDBC y SQL para obtener una lista de los tipos de datos compatibles.

Notas de uso:

- Una cláusula de declaración de iterador puede aparecer dentro de un programa Java en un lugar cualquiera en donde pueda existir una declaración de clase Java.
- Cuando una declaración de iterador de nombre contiene más de un par tipo de datos Java-ID de Java, todos los ID de Java de la lista deben ser exclusivos.

Tareas relacionadas:

- "Conexión a una fuente de datos utilizando SQLJ" en la página 349

Información relacionada:

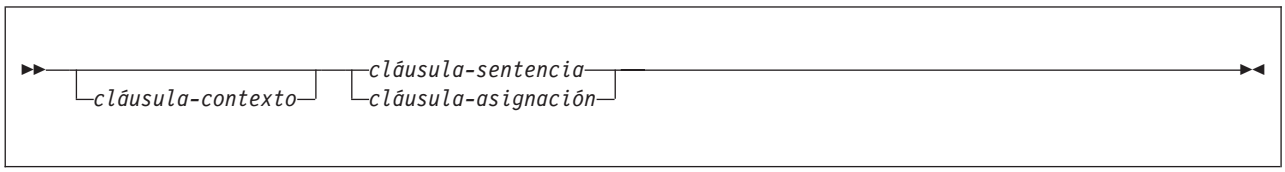
- "Cláusula implements de SQLJ" en la página 428
- "Cláusula with de SQLJ" en la página 429

Cláusula ejecutable de SQLJ

Una cláusula ejecutable contiene una sentencia de SQL o una sentencia de asignación. Una sentencia de asignación asigna el resultado de una operación de SQL a una variable Java.

El presente tema describe el formato general de una cláusula ejecutable.

Sintaxis:



Notas de uso:

- Una cláusula ejecutable puede aparecer dentro de un programa Java en un lugar cualquiera en donde pueda existir una sentencia de Java.
- SQLJ utiliza la clase `java.sql.SQLException` para notificar los códigos negativos de SQL resultantes de cláusulas ejecutables.

  Si SQLJ emite una excepción de ejecución durante la ejecución de una cláusula ejecutable, el valor de cualquier expresión de lenguaje principal de tipo OUT o INOUT no está definido.

Información relacionada:

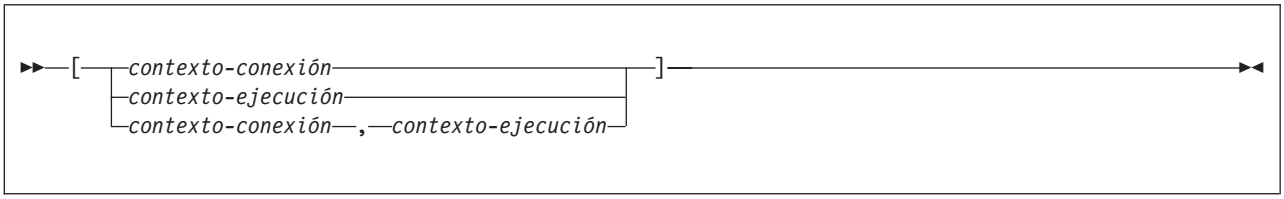
- "Cláusula de asignación de SQLJ" en la página 437
- "Cláusula context de SQLJ" en la página 434

- "Cláusula de sentencia de SQLJ" en la página 434

Cláusula context de SQLJ

La cláusula context especifica un contexto de conexión, un contexto de ejecución o ambas cosas. El contexto de conexión se utiliza para conectar con una fuente de datos. El contexto de ejecución se utiliza para supervisar y modificar la ejecución de sentencias de SQL.

Sintaxis:



Descripción:

contexto-conexión
Especifica un identificador Java válido que se ha declarado anteriormente en el programa de SQLJ. Este identificador debe estar declarado como instancia de la clase de contexto de conexión que SQLJ genera para una cláusula de declaración de conexión.

contexto-ejecución
Especifica un identificador Java válido que se ha declarado anteriormente en el programa de SQLJ. Este identificador debe estar declarado como instancia de la clase `sqlj.runtime.ExecutionContext`.

Notas de uso:

- Si no especifica un contexto de conexión en una cláusula ejecutable, SQLJ utiliza el contexto de conexión por omisión.
- Si no especifica un contexto de ejecución, SQLJ obtiene el contexto de ejecución a partir del contexto de conexión de la sentencia.

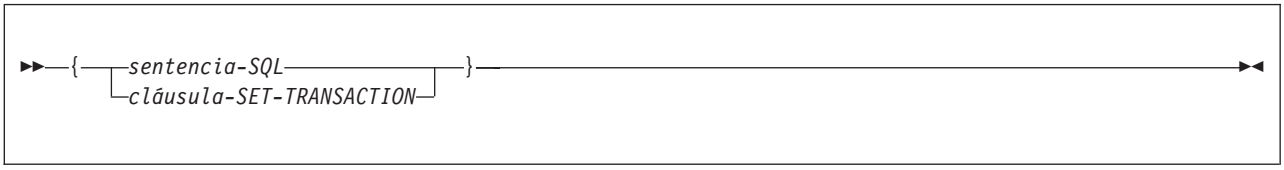
Tareas relacionadas:

- "Conexión a una fuente de datos utilizando SQLJ" en la página 349
- "Control de la ejecución de sentencias de SQL en SQLJ" en la página 381

Cláusula de sentencia de SQLJ

Una cláusula de sentencia contiene una sentencia de SQL o una cláusula SET TRANSACTION.

Sintaxis:



Descripción:

sentencia-SQL
Puede incluir sentencias de SQL de DB2 UDB para Linux, UNIX y Windows en una cláusula de sentencia. Consulte la Tabla 73.

cláusula-SET-TRANSACTION
Define el nivel de aislamiento de las sentencias de SQL del programa y la modalidad de acceso de la conexión. La cláusula SET TRANSACTION es equivalente a la sentencia SET TRANSACTION, que está descrita en el estándar de ANSI/ISO para SQL de 1992 y está soportada en algunas implementaciones de SQL. Consulte el tema Cláusula SET TRANSACTION de SQLJ para obtener más información.

Tabla 73. Sentencias de SQL válidas en una cláusula de sentencia de SQLJ

ALTER DATABASE
ALTER FUNCTION
ALTER INDEX
ALTER PROCEDURE
ALTER STOGROUP
ALTER TABLE
ALTER TABLESPACE
CALL
COMMENT ON
COMMIT
CREATE ALIAS
CREATE DATABASE
CREATE DISTINCT TYPE
CREATE FUNCTION
CREATE GLOBAL TEMPORARY TABLE
CREATE INDEX
CREATE PROCEDURE
CREATE STOGROUP
CREATE SYNONYM
CREATE TABLE
CREATE TABLESPACE
CREATE TRIGGER
CREATE VIEW
DECLARE GLOBAL TEMPORARY TABLE
DELETE
DROP ALIAS
DROP DATABASE
DROP DISTINCT TYPE
DROP FUNCTION
DROP INDEX
DROP PACKAGE
DROP PROCEDURE
DROP STOGROUP
DROP SYNONYM
DROP TABLE
DROP TABLESPACE
DROP TRIGGER
DROP VIEW
FETCH
GRANT
INSERT
LOCK TABLE
REVOKE
ROLLBACK

Tabla 73. Sentencias de SQL válidas en una cláusula de sentencia de SQLJ (continuación)

| SAVEPOINT
SELECT INTO
SET CURRENT DEFAULT TRANSFORM GROUP
SET CURRENT DEGREE
SET CURRENT EXPLAIN MODE
SET CURRENT EXPLAIN SNAPSHOT
SET CURRENT ISOLATION
SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION
SET CURRENT OPTIMIZATION HINT
SET CURRENT PACKAGESET (USER no está soportado)
SET CURRENT PRECISION
SET CURRENT QUERY OPTIMIZATION
SET CURRENT REFRESH AGE
SET CURRENT SCHEMA
SET PATH
UPDATE

Notas de uso:

- SQLJ da soporte a operaciones DELETE y UPDATE de posición y de búsqueda.
- Para una sentencia FETCH, una sentencia DELETE de posición o una sentencia UPDATE de posición, debe utilizar un iterador para apuntar a las filas de una tabla de resultados.

Tareas relacionadas:

- "Establecimiento del nivel de aislamiento para una transacción de SQLJ" en la página 354

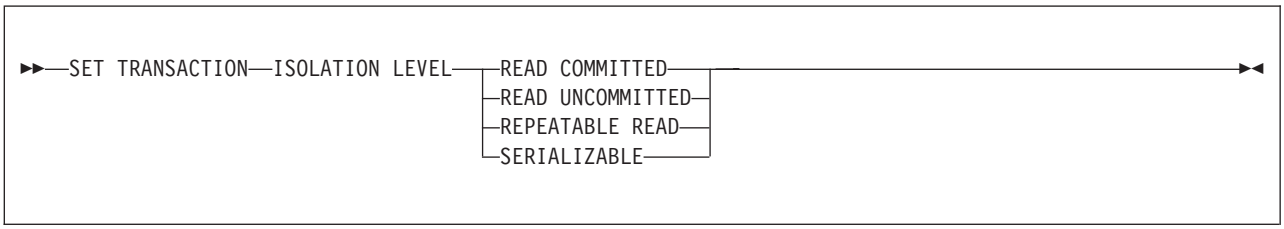
Información relacionada:

- "Cláusula SET TRANSACTION de SQLJ" en la página 436

Cláusula SET TRANSACTION de SQLJ

La cláusula SET TRANSACTION define el nivel de aislamiento de la unidad de trabajo actual.

Sintaxis:



Descripción:

ISOLATION LEVEL
Especifica uno de los niveles de aislamiento siguientes:

READ COMMITTED
Especifica que el nivel de aislamiento actual de DB2 es estabilidad del cursor.

READ UNCOMMITTED
Especifica que el nivel de aislamiento actual de DB2 es lectura no confirmada.

REPEATABLE READ
Especifica que el nivel de aislamiento actual de DB2 es estabilidad de lectura.

SERIALIZABLE
Especifica que el nivel de aislamiento actual de DB2 es lectura repetible.

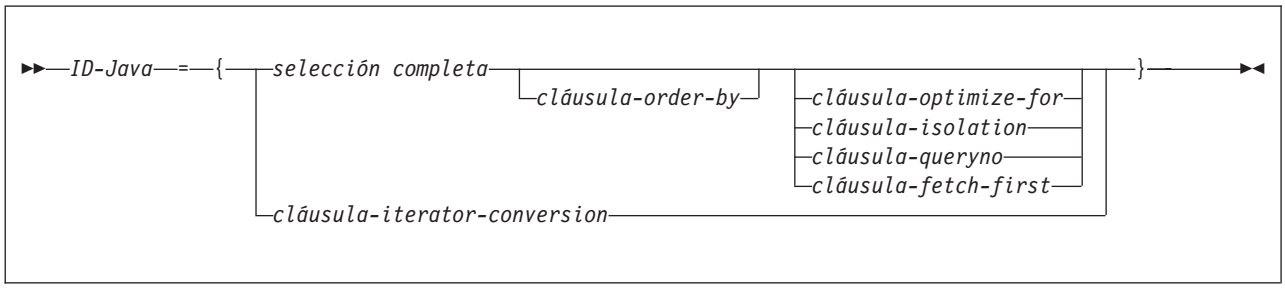
Notas de uso:

Puede ejecutar SET TRANSACTION solo al comienzo de una transacción.

Cláusula de asignación de SQLJ

La cláusula de asignación asigna el resultado de una operación de SQL a una variable.

Sintaxis:



Descripción:

ID-Java
Identifica un iterador que se declaró previamente como instancia de una clase de iterador.

selección completa
Genera una tabla de resultados.

cláusula-iterator-conversion
Consulte el tema Cláusula de conversión a iterador de SQLJ para obtener una descripción de esta cláusula.

Notas de uso:

- Si el objeto identificado por *ID-Java* es un iterador de posición, el número de columnas del conjunto de resultados debe coincidir con el número de columnas del iterador. Además, el tipo de datos de cada columna del conjunto de resultados debe ser compatible con el tipo de datos de la columna correspondiente del iterador. Consulte el tema Tipos de datos de Java, JDBC y SQL para obtener una lista de los tipos de datos compatibles de Java y SQL.
- Si el objeto identificado por *ID-Java* es un iterador de nombre, el nombre de cada método accesor debe coincidir con el nombre de una columna del conjunto de resultados, salvo en lo que respecta al uso de letras mayúsculas y minúsculas. Además, el tipo de datos del objeto devuelto por un método accesor debe ser compatible con el tipo de datos de la columna correspondiente del conjunto de resultados.
- Puede colocar una cláusula de asignación en un lugar cualquiera de un programa Java en donde puede existir una sentencia de asignación de Java. Para

no puede colocar una cláusula de asignación donde pueda haber una expresión de asignación de Java. Por ejemplo, no puede especificar una cláusula de asignación en la lista de control de una sentencia FOR.

Conceptos relacionados:

- "Uso de SQLJ y JDBC en la misma aplicación" en la página 373

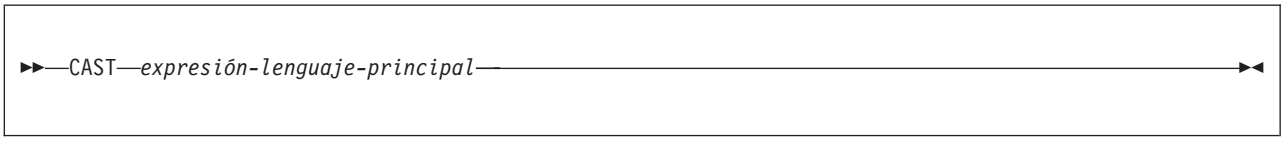
Información relacionada:

- "Selección completa" en la publicación *Consulta de SQL, Volumen 1*
- "Sentencia select" en la publicación *Consulta de SQL, Volumen 1*
- "Cláusula de conversión a iterador de SQLJ" en la página 438

Cláusula de conversión a iterador de SQLJ

La cláusula de conversión a iterador convierte un `conjunto de resultados` de JDBC en un iterador.

Sintaxis:



Descripción:

expresión-lenguaje-principal
Identifica el `conjunto de resultados` de JDBC que se debe convertir en un iterador de SQLJ.

Notas de uso:

- Si el iterador al que se debe convertir el `conjunto de resultados` es un iterador de posición, el número de columnas del `conjunto de resultados` debe coincidir con el número de columnas del iterador. Además, el tipo de datos de cada columna del `conjunto de resultados` debe ser compatible con el tipo de datos de la columna correspondiente del iterador.
- Si el iterador es un iterador de nombre, el nombre de cada método accesor debe coincidir con el nombre de una columna del `conjunto de resultados`, salvo en lo que respecta al uso de letras mayúsculas y minúsculas. Además, el tipo de datos del objeto devuelto por un método accesor debe ser compatible con el tipo de datos de la columna correspondiente del `conjunto de resultados`.
- Cuando se cierra un iterador que se ha generado mediante una cláusula de conversión a iterador, también se cierra el `conjunto de resultados` a partir del cual se creó el iterador.

Conceptos relacionados:

- "Uso de SQLJ y JDBC en la misma aplicación" en la página 373

Clases e interfaces selectas de sqlj.runtime

El paquete `sqlj.runtime` define las clases de tiempo de ejecución utilizadas por SQLJ. Este tema describe:

- Cada clase de `sqlj.runtime` que contiene métodos que puede invocar en sus programas de aplicación SQLJ

- Cada una de las interfaces que puede necesitar implementar en sus programas de aplicación SQLJ

Clase sqlj.runtime.ExecutionContext:

La clase `sqlj.runtime.ExecutionContext` se define para contextos de ejecución. Puede utilizar un contexto de ejecución para controlar la ejecución de sentencias de SQL. Después de declarar un contexto de ejecución y crear una instancia de ese contexto de ejecución, puede utilizar los métodos siguientes.

executeBatch

Formato:

```
public synchronized int[] executeBatch()
```

Ejecuta el lote de sentencias pendiente y devuelve una matriz de contajes de actualización. Si no existe ningún lote de sentencias pendiente, se devuelve un valor nulo. Cuando se invoca este método, se elimina el lote de sentencias, aunque la llamada produzca una excepción.

getBatchLimit

Formato:

```
synchronized public int getBatchLimit()
```

Devuelve el limite actual del lote, que es el número de sentencias que se añaden a un lote antes de ejecutarlo implícitamente.

getBatchUpdateCounts

Formato:

```
public synchronized int[] getBatchUpdateCounts()
```

Devuelve una matriz que contiene el número de filas actualizadas por cada sentencia que se ejecutó satisfactoriamente en un lote. Devuelve un valor nulo si ninguna sentencia del lote se ejecutó satisfactoriamente.

getMaxFieldSize

Formato:

```
public int getMaxFieldSize()
```

Proporciona el número máximo de bytes devueltos para una columna de caracteres o binaria cualquiera por una consulta que hace uso del contexto de ejecución existente. Un valor 0 significa que el número máximo de bytes es ilimitado.

getMaxRows

Formato:

```
public int getMaxRows()
```

Proporciona el número máximo de filas que son devueltas por una consulta cualquiera que hace uso del contexto de ejecución existente. Si el valor devuelto es 0, significa que el número máximo de filas es ilimitado.

getNextResultSet

Formatos:

```
public ResultSet getNextResultSet()
public ResultSet getNextResultSet(int actual)
```

Después de una llamada de procedimiento almacenado, devuelve un conjunto de resultados procedente del procedimiento almacenado. Si el valor devuelto es nulo, significa que no existen más conjuntos de resultados para devolver.

Cuando invoca `getNextResultSet()`, SQLJ cierra el conjunto de resultados que está abierto actualmente y pasa al conjunto de resultados siguiente. Cuando invoca `getNextResultSet(int actual)`, el valor de *actual* indica lo que SQLJ realiza con el conjunto de resultados abierto actualmente antes de pasar al conjunto de resultados siguiente:

java.sql.Statement.CLOSE_CURRENT_RESULT
Especifica que el objeto `ResultSet` actual se cierra cuando se devuelve el objeto `ResultSet` siguiente.

java.sql.Statement.KEEP_CURRENT_RESULT
Especifica que el objeto `ResultSet` actual permanece abierto cuando se devuelve el objeto `ResultSet` siguiente.

java.sql.Statement.CLOSE_ALL_RESULTS
Especifica que todos los objetos `ResultSet` abiertos se cierran cuando se devuelve el objeto `ResultSet` siguiente.

La ejecución de `getNextResultSet(int actual)` requiere utilizar JDK 1.4 o versión posterior.

getUpdateCount
Formato:

```
public abstract int getUpdateCount() throws SQLException
```

Devuelve:

ExecutionContext.ADD_BATCH_COUNT
Si la sentencia se añadió a un lote existente.

ExecutionContext.NEW_BATCH_COUNT
Si la sentencia era la primera sentencia de un nuevo lote.

ExecutionContext.EXEC_BATCH_COUNT
Si la sentencia era parte de un lote y el lote se ejecutó.

Otro valor entero
Si la sentencia se ejecutó en lugar de añadirla a un lote. Este valor es el número de filas que fueron actualizadas por la sentencia.

getWarnings
Formato:

```
public SQLWarning getWarnings()
```

Devuelve el primer aviso que fue notificado por la última operación de SQL que se ejecutó utilizando el contexto actual. Los avisos subsiguientes se encadenan al primer aviso.

Utilice este método para recuperar los SQLCODE positivos.

isBatching
Formato:

```
public synchronized boolean isBatching()
```

Devuelve el valor `true` si el proceso por lotes está habilitado. Devuelve el valor `false` si el proceso por lotes está inhabilitado.

setBatching
Formato:

```
public synchronized void setBatching(boolean)
```

Habilita o inhabilita el proceso por lotes.

setBatchLimit
Formato:

```
public synchronized void setBatchLimit(int)
```

Establece el número máximo de sentencias que se añaden a un lote antes de ejecutarlo implícitamente. Los valores posibles para el parámetro de entrada son:

ExecutionContext.UNLIMITED_BATCH
Indica que la ejecución implícita solo se produce cuando SQLJ encuentra una sentencia que es procesable por lotes pero incompatible, o que no es procesable por lotes. Establecer este valor es lo mismo que no invocar `setBatchLimit`.

ExecutionContext.AUTO_BATCH
Indica que la ejecución implícita se produce cuando el número de sentencias del lote alcanza un valor definido por SQLJ.

Entero positivo
Es el número de sentencias que se añaden al lote antes de que SQLJ ejecute el lote implícitamente. El lote puede ejecutarse antes de que se haya añadido este número de sentencias si SQLJ encuentra una sentencia que es procesable por lotes pero incompatible, o que no es procesable por lotes.

setMaxFieldSize
Formato:

```
public void setMaxFieldSize(int max)
```

Especifica el número máximo de bytes que son devueltos para una columna de caracteres o binaria cualquiera por una consulta que hace uso del contexto de ejecución existente. El valor por omisión es 0, que significa que el número máximo de bytes devueltos es ilimitado.

setMaxRows
Formato:

```
public void setMaxRows(int max)
```

Especifica el número máximo de filas que son devueltas por una consulta cualquiera que hace uso del contexto de ejecución existente. El valor por omisión es 0, que significa que el número máximo de filas devueltas es ilimitado.

sqlj.runtime.ConnectionContext interface:

`sqlj.runtime.ConnectionContext` es una interfaz que SQLJ implementa cuando se ejecuta una cláusula de declaración de conexión y por tanto se crea una clase de contexto de conexión.

Suponga que declara una conexión llamada Ctx. Puede entonces utilizar los métodos siguientes para definir o cambiar el contexto por omisión.

getDefaultContext
Formato:

```
public static Ctx getDefaultContext()
```

Devuelve el objeto del contexto de conexión por omisión correspondiente a la clase Ctx.

SetDefaultContext
Formato:

```
public static void Ctx setDefaultContext(Ctx contexto-por-omisión)
```

Define el objeto del contexto de conexión por omisión correspondiente a la clase Ctx.

Interfaz sqlj.runtime.ForUpdate:

Implementa la interfaz `sqlj.runtime.ForUpdate` para operaciones UPDATE o DELETE de posición. La interfaz `sqlj.runtime.ForUpdate` se implementa en una cláusula de declaración de iterador de SQLJ.

Interfaz sqlj.runtime.NamedIterator:

`sqlj.runtime.NamedIterator` es una interfaz que SQLJ implementa cuando se declara un iterador de nombre. Cuando se declara una instancia de un iterador de nombre, SQLJ crea un método accesor para cada columna de la tabla de resultados prevista. Un método accesor devuelve los datos contenidos en las columnas de la tabla de resultados. El nombre de un método accesor coincide con el nombre de la columna correspondiente del iterador de nombre.

Además de los métodos accesores, SQLJ genera los métodos siguientes que el usuario puede invocar en su aplicación SQLJ.

close
Formato:

```
public abstract void close() throws SQLException
```

Libera recursos de base de datos que el iterador está utilizando.

isClosed
Formato:

```
public abstract boolean isClosed() throws SQLException
```

Devuelve el valor `true` si se invocado el método `close`.

next
Formato:

```
public abstract boolean next() throws SQLException
```

Avanza el iterador hasta la fila siguiente. Antes de invocar por primera vez una instancia del método `next`, el iterador se posiciona antes de la primera fila de la tabla de resultados. `next` devuelve el valor `true` cuando está disponible una fila siguiente, y el valor `false` cuando se han recuperado todas las filas.

Interfaz sqlj.runtime.PositionedIterator:

`sqlj.runtime.PositionedIterator` es una interfaz que SQLJ implementa cuando se declara un iterador de posición. Después de declarar y crear una instancia de un iterador de posición, puede utilizar el método siguiente.

endFetch

Formato:

```
public abstract boolean endFetch() throws SQLException
```

Devuelve el valor `true` si el iterador no está posicionado en una fila.

Interfaz sqlj.runtime.ResultSetIterator:

`sqlj.runtime.ResultSetIterator` es una interfaz que SQLJ implementa cuando se declara un iterador. Después de declarar y crear una instancia de un iterador, puede utilizar los métodos siguientes.

clearWarnings

Formato:

```
public abstract void clearWarnings() throws SQLException
```

Devuelve un valor nulo hasta que se notifica un nuevo aviso para el iterador.

close

Formato:

```
public abstract void close() throws SQLException
```

Libera recursos de base de datos que el iterador está utilizando.

getResultSet

Formato:

```
public abstract ResultSet getResultSet() throws SQLException
```

Devuelve una representación en forma de conjunto de resultados de JDBC para un iterador SQLJ.

getWarnings

Formato:

```
public abstract SQLWarning getWarnings() throws SQLException
```

Devuelve el primer aviso que es notificado por llamadas al iterador. Los avisos de iterador subsiguientes se encadenan a este aviso de SQL. La cadena de avisos se elimina automáticamente cada vez que se lee una nueva fila.

isClosed

Formato:

```
public abstract boolean isClosed() throws SQLException
```

Devuelve el valor `true` si se invocado el método `close`.

next

Formato:

```
public abstract boolean next() throws SQLException
```

Avanza el iterador hasta la fila siguiente. Antes de invocar por primera vez una instancia del método `next`, el iterador se posiciona antes de la primera fila de la tabla de resultados. `next` devuelve el valor `true` cuando está disponible una fila siguiente, y el valor `false` cuando se han recuperado todas las filas.

Interfaz sqlj.runtime.Scrollable:

`sqlj.runtime.Scrollable` es una interfaz que se implementa al declarar un iterador desplazable. Utilice los métodos `sqlj.runtime.Scrollable` para desplazarse por la tabla de resultados y comprobar su posición en ella.

absolute(int)

Formato:

```
public abstract boolean absolute (int n) throws SQLException
```

Desplaza el cursor hasta una fila especificada.

Si *n*>0, sitúa el iterador en la fila *n* de la tabla de resultados. Si *n*<0 y *m* es el número de filas de la tabla de resultados, sitúa el iterador en la fila *m+n+1* de la tabla de resultados.

Si el valor absoluto de *n* es mayor que el número de filas de la tabla de resultados, sitúa el cursor después de la última fila si *n* es negativo, o antes de la primera fila si *n* es positivo.

`Absolute(1)` es lo mismo que `first()`. `Absolute(-1)` es lo mismo que `last()`.

Devuelve el valor `true` si el iterador está en una fila. En otro caso, devuelve el valor `false`.

afterLast()

Formato:

```
public abstract void afterLast() throws SQLException
```

Coloca el iterador después de la última fila de la tabla de resultados.

beforeFirst()

Formato:

```
public abstract void beforeFirst() throws SQLException
```

Coloca el iterador antes de la primera fila de la tabla de resultados.

first()

Formato:

```
public abstract boolean first() throws SQLException
```

Coloca el iterador en la primera fila de la tabla de resultados.

Devuelve el valor `true` si el iterador está en una fila. En otro caso, devuelve el valor `false`.

getFetchDirection()

Formato:

```
public abstract int getFetchDirection ( ) throws SQLException
```

Devuelve la dirección de búsqueda del iterador. Los valores posibles son:

sqlj.runtime.ResultSetIterator.FETCH_FORWARD
Las filas se procesan en dirección de avance, desde la primera a la última.

sqlj.runtime.ResultSetIterator.FETCH_REVERSE
Las filas se procesan en dirección inversa, desde la última a la primera.

sqlj.runtime.ResultSetIterator.FETCH_UNKNOWN
El orden de proceso es desconocido.

isAfterLast()

Formato:

```
public abstract boolean isAfterLast() throws SQLException
```

Devuelve el valor `true` si el iterador está situado después de la última fila de la tabla de resultados. En otro caso, devuelve el valor `false`.

isBeforeFirst()

Formato:

```
public abstract boolean isBeforeFirst() throws SQLException
```

Devuelve el valor `true` si el iterador está situado antes de la primera fila de la tabla de resultados. En otro caso, devuelve el valor `false`.

isFirst()

Formato:

```
public abstract boolean isFirst() throws SQLException
```

Devuelve el valor `true` si el iterador está situado en la primera fila de la tabla de resultados. En otro caso, devuelve el valor `false`.

isLast()

Formato:

```
public abstract boolean isLast() throws SQLException
```

Devuelve el valor `true` si el iterador está situado en la última fila de la tabla de resultados. En otro caso, devuelve el valor `false`.

last()

Formato:

```
public abstract boolean last() throws SQLException
```

Coloca el iterador en la última fila de la tabla de resultados.

Devuelve el valor `true` si el iterador está en una fila. En otro caso, devuelve el valor `false`.

previous()

Formato:

```
public abstract boolean previous() throws SQLException
```

Coloca el iterador en la fila anterior de la tabla de resultados.

Devuelve el valor `true` si el iterador está en una fila. En otro caso, devuelve el valor `false`.

relative(int)

Formato:

```
public abstract boolean relative(int n) throws SQLException
```

Si *n*>0, sitúa el iterador en la fila que está situada *n* filas después de la fila actual. Si *n*<0, sitúa el iterador en la fila que está situada *n* filas antes de la fila actual. Si *n*=0, sitúa el iterador en la fila actual.

El cursor debe estar en una fila válida de la tabla de resultados para poder utilizar este método. Si el cursor está antes de la primera fila o después de la última fila, el método emite una `excepción de SQL`.

Suponga que *m* es el número de filas de la tabla de resultados y *x* es el número de fila actual de la tabla de resultados. Si *n*>0 y *x*+*n*>*m*, el iterador se sitúa después de la última fila. Si *n*<0 y *x*+*n*<1, el iterador se sitúa antes de la primera fila.

Devuelve el valor `true` si el iterador está en una fila. En otro caso, devuelve el valor `false`.

setFetchDirection(int)

Formato:

```
public abstract void setFetchDirection (int) throws SQLException
```

Indica al entorno de ejecución de SQLJ la dirección en la que se procesan las filas del objeto iterador. Los valores posibles son:

sqlj.runtime.ResultSetIterator.FETCH_FORWARD
Las filas se procesan en dirección de avance, desde la primera a la última.

sqlj.runtime.ResultSetIterator.FETCH_REVERSE
Las filas se procesan en dirección inversa, desde la última a la primera.

sqlj.runtime.ResultSetIterator.FETCH_UNKNOWN
El orden de proceso es desconocido.

Tareas relacionadas:

- "Realización de actualizaciones por lotes en aplicaciones SQLJ" en la página 383
- "Conexión a una fuente de datos utilizando SQLJ" en la página 349
- "Utilización de un iterador de nombre en una aplicación SQLJ" en la página 358
- "Utilización de un iterador de posición en una aplicación SQLJ" en la página 361
- "Ejecución de operaciones UPDATE y DELETE de posición en una aplicación SQLJ" en la página 363
- "Utilización de iteradores desplazables en una aplicación SQLJ" en la página 389
- "Manejo de avisos de SQL en una aplicación SQLJ" en la página 372
- "Control de la ejecución de sentencias de SQL en SQLJ" en la página 381

Información de consulta sobre el Controlador JDBC universal de DB2

Las secciones siguientes contienen información que es específica del Controlador JDBC universal de DB2.

Resumen de las extensiones del Controlador JDBC de DB2 Universal para JDBC

Este tema describe las API de JDBC que son específicas del Controlador JDBC universal de DB2.

Para utilizar cualquiera de los métodos descritos en este tema, debe convertir una instancia de la clase estándar asociada de JDBC en una instancia de la clase específica de DB2. Por ejemplo:

```
javax.sql.DataSource ds =
 new com.ibm.db2.jcc.DB2SimpleDataSource();
((com.ibm.db2.jcc.DB2BaseDataSource) ds).setServerName("sysmvs1.stl.ibm.com");
```

Clase DB2ActiveServerList:

La clase com.ibm.db2.jcc.DB2ActiveServerList implementa las interfaces java.io.Serializable y javax.naming.Referenceable.

Métodos de DB2ActiveServerList:

getAlternatePortNumber
Formato:

```
public int[] getAlternatePortNumber()
```

Obtiene los números de puerto que están asociados a los servidores DB2 UDB alternativos.

getAlternateServerName
Formato:

```
public String[] getAlternateServerName()
```

Obtiene una matriz que contiene los nombres de los servidores DB2 UDB alternativos. Estos valores son direcciones IP o nombres de servidor DNS.

setAlternatePortNumber
Formato:

```
public void setAlternatePortNumber(int[] alternatePortNumberList)
```

Define los números de puerto que están asociados a los servidores DB2 UDB alternativos.

setAlternateServerName
Formato:

```
public void setAlternateServerName(String[] alternateServer)
```

Define los nombres de servidor alternativo para los servidores DB2 UDB. Estos valores son direcciones IP o nombres de servidor DNS.

Clase DB2BaseDataSource:

La clase com.ibm.db2.jcc.DB2BaseDataSource es la clase padre de fuente de datos abstracta para todas las implementaciones de javax.sql.DataSource, javax.sql.ConnectionPoolDataSource y javax.sql.XADataSource específicas de DB2.

Propiedades de DB2BaseDataSource:

Las propiedades siguientes se definen solamente para el Controlador JDBC universal de DB2. Consulte el tema Propiedades del Controlador JDBC de DB2 Universal para obtener descripciones de estas propiedades.

Cada una de estas propiedades tiene un método setXXX para definir el valor de la propiedad y un método getXXX para obtener el valor. Un método setXXX tiene este formato:

```
void setNombre_propiedad(tipo_datos valor_propiedad)
```

Un método getXXX tiene este formato:

```
tipo_datos getnombre_propiedad()
```

Nombre_propiedad es el nombre de propiedad no calificado, con el primer carácter escrito en mayúsculas.

La Tabla 74 lista las propiedades del Controlador JDBC universal de DB2 y los tipos de datos asociados a ellas.

Tabla 74. Propiedades del Controlador JDBC universal de DB2 y tipos de datos asociados a ellas

Nombre de propiedad	Tipo de datos
com.ibm.db2.jcc.DB2BaseDataSource.activeServerListJNDIName	String
com.ibm.db2.jcc.DB2BaseDataSource.clientAccountingInformation	String
com.ibm.db2.jcc.DB2BaseDataSource.clientApplicationInformation	String
com.ibm.db2.jcc.DB2BaseDataSource.clientUser	String
com.ibm.db2.jcc.DB2BaseDataSource.clientWorkstation	String
com.ibm.db2.jcc.DB2BaseDataSource.cliSchema	String
com.ibm.db2.jcc.DB2BaseDataSource.currentFunctionPath	String
com.ibm.db2.jcc.DB2BaseDataSource.currentLockTimeout	int
com.ibm.db2.jcc.DB2BaseDataSource.currentPackagePath	String
com.ibm.db2.jcc.DB2BaseDataSource.cursorSensitivity	int
com.ibm.db2.jcc.DB2BaseDataSource.currentSchema	String
com.ibm.db2.jcc.DB2BaseDataSource.currentSQLID	String
com.ibm.db2.jcc.DB2BaseDataSource.currentSQLID	String
com.ibm.db2.jcc.DB2BaseDataSource.databaseName	String
com.ibm.db2.jcc.DB2BaseDataSource.deferPrepares	boolean
com.ibm.db2.jcc.DB2BaseDataSource.description	String
com.ibm.db2.jcc.DB2BaseDataSource.driverType	int
com.ibm.db2.jcc.DB2BaseDataSource.fullyMaterializeLobData	boolean
com.ibm.db2.jcc.DB2BaseDataSource.gssCredential	Object
com.ibm.db2.jcc.DB2BaseDataSource.jdbcCollection	String
com.ibm.db2.jcc.DB2BaseDataSource.keepDynamic	int
com.ibm.db2.jcc.DB2BaseDataSource.kerberosServerPrincipal	String
com.ibm.db2.jcc.DB2BaseDataSource.logWriter	PrintWriter
com.ibm.db2.jcc.DB2BaseDataSource.portNumber	int
com.ibm.db2.jcc.DB2BaseDataSource.resultSetHoldability	int
com.ibm.db2.jcc.DB2BaseDataSource.securityMechanism	int
com.ibm.db2.jcc.DB2BaseDataSource.serverName	String
com.ibm.db2.jcc.DB2BaseDataSource.readOnly	boolean
com.ibm.db2.jcc.DB2BaseDataSource.traceFile	String
com.ibm.db2.jcc.DB2BaseDataSource.traceLevel	int
com.ibm.db2.jcc.DB2BaseDataSource.user	String

Métodos de DB2BaseDataSource:

Además de los métodos `getXXX` y `setXXX` para las propiedades de DB2BaseDataSource, están definidos los métodos siguientes solo para el Controlador JDBC universal de DB2.

getReference

Formato:

```
public javax.naming.Reference getReference()
  throws javax.naming.NamingException
```

Obtiene la referencia (`Reference`) de un objeto `DataSource`. Para obtener una explicación sobre `Reference`, consulte la descripción de `javax.naming.Referenceable` en la documentación de JNDI, que se encuentra en:

`http://java.sun.com/products/jndi/docs.html`

Interfaz DB2Connection:

La interfaz `com.ibm.db2.jcc.DB2Connection` amplía la interfaz `java.sql.Connection`.

Métodos de DB2Connection:

Los métodos siguientes se definen solamente para el Controlador JDBC universal de DB2.

getDB2ClientAccountingInformation

Formato:

```
public String getDB2ClientAccountingInformation()
    throws SQLException
```

Obtiene información contable para el cliente actual.

getDB2ClientApplicationInformation

Formato:

```
public String getDB2ClientApplicationInformation()
    throws SQLException
```

Obtiene información de aplicación para el cliente actual.

getDB2ClientUser

Formato:

```
public String getDB2ClientUser()
    throws SQLException
```

Obtiene el nombre de usuario del cliente actual de la conexión. Este nombre no es el valor de usuario (user) de la conexión JDBC.

getDB2ClientWorkstation

Formato:

```
public String getDB2ClientWorkstation()
    throws SQLException
```

Obtiene el nombre de estación de trabajo del cliente actual.

getDB2CurrentPackagePath

Formato:

```
public String getDB2CurrentPackagePath()
    throws SQLException
```

Obtiene la lista de colecciones de paquetes DB2 en donde se buscan los paquetes de Controlador JDBC universal de DB2.

getDB2CurrentPackageSet

Formato:

```
public String getDB2CurrentPackageSet()
    throws SQLException
```

Obtiene el ID de colección de la conexión.

getDB2SystemMonitor

Formato:

```
public abstract DB2SystemMonitor getDB2SystemMonitor()
    throws SQLException
```

Obtiene el objeto supervisor del sistema de la conexión. Cada conexión del Controlador JDBC universal de DB2 puede tener un supervisor del sistema individual. Vea el tema "Interfaz DB2SystemMonitor" en la página 456 para obtener más información.

getJccLogWriter

Formato:

```
public PrintWriter getJccLogWriter()
    throws SQLException
```

Obtiene el destino de rastreo actual para la función de rastreo del Controlador JDBC universal de DB2.

setDB2ClientAccountingInformation

Formato:

```
public void setDB2ClientAccountingInformation(String info)
    throws SQLException
```

Especifica información contable para la conexión. Esta información se utiliza con fines de contabilidad de clientes. Este valor puede cambiar durante una conexión.

Descripción de parámetro:

info
Información contable especificada por el usuario. La longitud máxima depende del servidor. Para un servidor DB2 UDB para Linux, UNIX y Windows, la longitud máxima es 255 bytes. Es válido especificar una serie vacía Java ("") para este valor de parámetro, pero no es válido un valor `nulo` Java.

setDB2ClientApplicationInformation

Formato:

```
public void setDB2ClientApplicationInformation(String info)
    throws SQLException
```

Especifica información de aplicación para la conexión. Esta información se utiliza con fines de contabilidad de clientes. Este valor puede cambiar durante una conexión.

Descripción de parámetro:

info
Información de aplicación especificada por el usuario. La longitud máxima depende del servidor. Para un servidor DB2 UDB para Linux, UNIX y

Windows, la longitud máxima es 255 bytes. Es válido especificar una serie vacía Java ("") para este valor de parámetro, pero no es válido un valor `nulo` Java.

setDB2ClientUser

Formato:

```
public void setDB2ClientUser(String user)
    throws SQLException
```

Especifica el nombre de usuario del cliente actual de la conexión. Este nombre se utiliza con fines de contabilidad de clientes, y no es el valor de usuario (user) de la conexión JDBC. A diferencia del valor de usuario de la conexión JDBC, el nombre de usuario del cliente actual puede cambiar durante una conexión.

Descripción de parámetro:

user
Es el ID de usuario del cliente actual. La longitud máxima depende del servidor. Para un servidor DB2 UDB para Linux, UNIX y Windows, la longitud máxima es 255 bytes. Es válido especificar una serie vacía Java ("") para este valor de parámetro, pero no es válido un valor `nulo` Java.

setDB2ClientWorkstation

Formato:

```
public void setDB2ClientWorkstation(String nombre)
    throws SQLException
```

Especifica el nombre de estación de trabajo del cliente actual de la conexión. Este nombre se utiliza con fines de contabilidad de clientes. El nombre de estación de trabajo del cliente actual puede cambiar durante una conexión.

Descripción de parámetro:

nombre
Es el nombre de estación de trabajo del cliente actual. La longitud máxima depende del servidor. Para un servidor DB2 UDB para Linux, UNIX y Windows, la longitud máxima es 255 bytes. Es válido especificar una serie vacía Java ("") para este valor de parámetro, pero no es válido un valor `nulo` Java.

setDB2CurrentPackagePath

Formato:

```
public void setDB2CurrentPackagePath(String packagePath)
    throws SQLException
```

Especifica una lista de los ID de colección en los que DB2 busca los paquetes DB2 del Controlador JDBC universal de DB2.

Descripción de parámetro:

packagePath
Es una lista de los ID de colección, separados por comas.

setDB2CurrentPackageSet

Formato:

```
public void setDB2CurrentPackageSet(String packageSet)
    throws SQLException
```

Especifica el ID de colección de la conexión. Cuando define este valor, también define el ID de colección de la instancia del Controlador JDBC universal de DB2 que se utiliza para la conexión.

Descripción de parámetro:

packageSet
Es el ID de colección de la conexión. La longitud máxima del valor de packageSet es 18 bytes. Puede invocar este método como alternativa a ejecutar la sentencia SET CURRENT PACKAGESET de SQL en su programa.

setJccLogWriter
Formatos:

```
public void setJccLogWriter(PrintWriter logWriter)
    throws SQLException

public void setJccLogWriter(PrintWriter logWriter, int traceLevel)
    throws SQLException
```

Habilita o inhabilita la función de rastreo del Controlador JDBC universal de DB2, o cambia el destino de rastreo durante una conexión activa.

Descripciones de parámetros:

logWriter
Es un objeto de tipo `java.io.PrintWriter` en donde el Controlador JDBC universal de DB2 escribe los datos de salida de rastreo. Para desactivar el rastreo, establezca el valor de *logWriter* en `null`.

traceLevel
Especifica los tipos de rastreos que se deben recoger. Consulte la descripción de la propiedad traceLevel en el tema Propiedades del Controlador JDBC de DB2 Universal para conocer los valores válidos.

Interfaz DB2DatabaseMetaData:

La interfaz `com.ibm.db2.jcc.DB2DatabaseMetaData` amplía la interfaz `java.sql.DatabaseMetaData`.

Métodos de DB2DatabaseMetaData:

Los métodos siguientes se definen solamente para el Controlador JDBC universal de DB2.

Interfaz DB2Diagnosable:

La interfaz `com.ibm.db2.jcc.DB2Diagnosable` proporciona un mecanismo para obtener información de diagnóstico de DB2 a partir de una excepción `SQLException` de DB2.

Métodos de DB2Diagnosable:

Los métodos siguientes se definen solamente para el Controlador JDBC universal de DB2.

getSqlca
Formato:

```
public DB2Sqlca getSqlca()
```

Devuelve un objeto DB2Sqlca a partir de una excepción java.sql.Exception que se produce al utilizar un Controlador JDBC universal de DB2.

getThrowable

Formato:

```
public Throwable getThrowable()
```

Devuelve un objeto java.lang.Throwable a partir de una excepción java.sql.Exception que se produce al utilizar un Controlador JDBC universal de DB2.

printTrace

Formato:

```
static public void printTrace(java.io.PrintWriter printWriter,
  String header)
```

Imprime información de diagnóstico después de emitirse una excepción java.sql.Exception al utilizar un Controlador JDBC universal de DB2.

Descripciones de parámetros:

printWriter
Es el destino de la información de diagnóstico.

header
Es información definida por el usuario que se imprime al comienzo de los datos de salida.

Clase DB2ExceptionFormatter:

La clase com.ibm.db2.jcc.DB2ExceptionFormatter contiene métodos para imprimir información de diagnóstico en una corriente de datos.

Métodos de DB2ExceptionFormatter:

Los métodos siguientes se definen solamente para el Controlador JDBC universal de DB2.

printTrace

Formatos:

```
static public void printTrace(java.sql.SQLException sqlException,
  java.io.PrintWriter printWriter, String header)

static public void printTrace(DB2Sqlca sqlca,
  java.io.PrintWriter printWriter, String header)

static public void printTrace(java.lang.Throwable throwable,
  java.io.PrintWriter printWriter, String header)
```

Imprime información de diagnóstico después de emitirse una excepción.

Descripciones de parámetros:

sqlException | sqlca | throwable
Es la excepción que se emitió durante una operación anterior de JDBC o Java.

printWriter
Es el destino de la información de diagnóstico.

header
Es información definida por el usuario que se imprime al comienzo de los datos de salida.

Interfaz DB2RowID:

La clase `com.ibm.db2.jcc.DB2RowID` se utiliza para declarar objetos Java a fin de utilizarlos con el tipo de datos ROWID de DB2.

Métodos de DB2RowID:

El método siguiente se define solamente para el Controlador JDBC universal de DB2.

getBytes
Formato:

```
public byte[] getBytes()
```

Convierte un objeto com.ibm.jcc.DB2RowID a bytes.

Clase DB2SimpleDataSource:

La clase `com.ibm.db2.jcc.DB2SimpleDataSource` amplía la clase `DataBaseDataSource`. Un objeto `DataBaseDataSource` no da soporte a la agrupación de conexiones ni a las transacciones distribuidas. Ese objeto contiene todas las propiedades y métodos que están contenidos en la clase `DB2BaseDataSource`. Además, `DB2SimpleDataSource` contiene las siguientes propiedades específicas del Controlador JDBC universal de DB2.

Propiedades de DB2SimpleDataSource:

La propiedad siguiente se define solamente para el Controlador JDBC universal de DB2. Consulte el tema Propiedades del Controlador JDBC de DB2 Universal para obtener una explicación de esta propiedad.
String com.ibm.db2.jcc.DB2SimpleDataSource.password

Métodos de DB2SimpleDataSource:

El método siguiente se define solamente para el Controlador JDBC universal de DB2.

setPassword
Formato:

```
public void setPassword( String password )
```

Define la contraseña para el objeto `DB2SimpleDataSource`. No existe un método correspondiente para `getPassword`. Por tanto, la contraseña no se puede cifrar, pues no hay forma de recuperar la contraseña para poder descifrarla.

Clase DB2Sqlca:

La clase `com.ibm.db2.jcc.DB2Sqlca` es una encapsulación de la SQLCA de DB2. .

Métodos de DB2Sqlca:

Los métodos siguientes se definen solamente para el Controlador JDBC universal de DB2.

getMessage
Formato:

```
public abstract String getMessage()
```

Devuelve el texto de mensajes de error.

getSqlCode
Formato:

```
public abstract int getSqlCode()
```

Devuelve el valor de códigos de error de SQL.

getSqlErrd
Formato:

```
public abstract int[] getSqlErrd()
```

Devuelve una matriz, cuyos elementos contienen un SQLERRD de SQLCA.

getSqlErrmc
Formato:

```
public abstract String getSqlErrmc()
```

Devuelve una cadena de caracteres que contiene los valores SQLERRMC de SQLCA, delimitados por espacios.

getSqlErrmcTokens
Formato:

```
public abstract String[] getSqlErrmcTokens()
```

Devuelve una matriz, cuyos elementos contienen un símbolo SQLERRMC de SQLCA.

getSqlErrd
Formato:

```
public abstract int[] getSqlErrd()
```

Devuelve una matriz, cuyos elementos contienen un valor SQLERRP de SQLCA.

getSqlErrp
Formato:

```
public abstract String getSqlErrp()
```

Devuelve el valor SQLERRP de SQLCA.

getSqlState
Formato:

```
public abstract String getSqlState()
```

Devuelve el valor SQLSTATE de SQLCA.

getSqlWarn
Formato:

```
public abstract char[] getSqlWarn()
```

Devuelve una matriz, cuyos elementos contienen un valor SQLWARN de SQLCA.

Interfaz DB2SystemMonitor:

La interfaz `com.ibm.db2.jcc.DB2SystemMonitor` se utiliza para recoger datos de supervisión del sistema correspondientes a una conexión. Cada conexión puede tener una sola instancia de `DB2SystemMonitor`.

Campos de DB2SystemMonitor:

Los campos siguientes se definen solamente para el Controlador JDBC universal de DB2.

public final static int RESET_TIMES
public final static int ACCUMULATE_TIMES
Estos valores son argumentos del método `DB2SystemMonitor.start`. RESET_TIMES inicializa a cero los contadores de tiempo antes del comienzo de la supervisión. ACCUMULATE_TIMES no pone a cero los contadores de tiempo.

Métodos de DB2SystemMonitor:

Los métodos siguientes se definen solamente para el Controlador JDBC universal de DB2.

enable
Formato:

```
public void enable(boolean on)
  throws java.sql.SQLException
```

Habilita el supervisor del sistema que está asociado a una conexión. Este método no se puede invocar durante la supervisión. Todos los tiempo se inicializan cuando se invoca `enable`.

getApplicationTimeMillis
Formato:

```
public long getApplicationTimeMillis()
  throws java.sql.SQLException
```

Devuelve la suma de los tiempos transcurridos correspondientes a la aplicación, el controlador JDBC, la E/S de red y el servidor DB2. El tiempo está expresado en milisegundos.

Un intervalo de tiempo transcurrido supervisado es la diferencia, en milisegundos, entre estos puntos en el proceso del controlador JDBC:

Inicio de intervalo
Es el momento en el que se invoca `start`.

Fin de intervalo
Es el momento en el que se invoca `stop`.

`getApplicationTimeMillis` devuelve un 0 si la supervisión del sistema está inhabilitada. Si se invoca este método sin invocar primero `stop`, se produce una excepción `SQLException`.

getCoreDriverTimeMicros
Formato:

```
public long getCoreDriverTimeMicros()
  throws java.sql.SQLException
```

Devuelve la suma de los tiempos transcurridos de API supervisada que se recogieron mientras la supervisión del sistema estaba habilitada. El tiempo está expresado en microsegundos.

Una API supervisada es un método de controlador JDBC para la cual se obtiene el tiempo de proceso. En general, los tiempos transcurridos solo se supervisan para las API que puedan producir interacción con la E/S de red o servidor DB2. Por ejemplo, los métodos `PreparedStatement.setXXX` y y `ResultSet.getXXX` no se supervisan.

El tiempo transcurrido de las API supervisadas incluye el tiempo total que se invierte en el controlador para una llamada de método. Este tiempo incluye cualquier tiempo de E/S de red y tiempo transcurrido de servidor DB2.

Un intervalo de tiempo transcurrido de API supervisada es la diferencia, en microsegundos, entre estos puntos en el proceso del controlador JDBC:

Inicio de intervalo
Es el momento en el que la aplicación llama a la API supervisada.

Fin de intervalo
Inmediatamente antes de este momento la API supervisada devuelve el control a la aplicación.

`getCoreDriverTimeMicros` devuelve un 0 si la supervisión del sistema está inhabilitada. Si se invoca este método sin primero llamar al método `stop`, o si se invoca este método cuando la JVM subyacente no da soporte a la notificación de tiempos en microsegundos, se produce una excepción `SQLException`.

getNetworkIOTimeMicros
Formato:

```
public long getNetworkIOTimeMicros()
  throws java.sql.SQLException
```

Devuelve la suma de los tiempos transcurridos de E/S de red que se recogieron mientras la supervisión del sistema estaba habilitada. El tiempo está expresado en microsegundos.

El tiempo transcurrido de E/S de red incluye el tiempo que se invierte en escribir y leer datos de DRDA procedentes de corrientes de E/S de la red. Un intervalo de tiempo transcurrido de E/S de red es el intervalo de tiempo que se invierte en realizar las operaciones siguientes en el controlador JDBC:

- Emitir un mandato de TCP/IP para enviar un mensaje de DRDA al servidor DB2. Este intervalo de tiempo es la diferencia, en microsegundos, entre los momentos inmediatamente anterior y posterior a la realización de una escritura y vaciado en la corriente de E/S de la red.
- Emitir un mandato de TCP/IP para recibir mensajes de respuesta de DRDA procedentes del servidor DB2. Este intervalo de tiempo es la diferencia, en microsegundos, entre los momentos inmediatamente anterior y posterior a la realización de una lectura en la corriente de E/S de la red.

Los intervalos de tiempo de E/S de red se recogen para todas las operaciones de envío y recepción, incluido el envío de mensajes para operaciones de confirmación y retrotracción.

El tiempo invertido en la espera a causa de operaciones de E/S de red puede verse afectado por retrasos en la CPU que despacha peticiones de SQL de baja prioridad en el servidor DB2. Los intervalos de tiempo de E/S de red incluyen el tiempo transcurrido del servidor DB2.

`getNetworkIOTimeMicros` devuelve un 0 si la supervisión del sistema está inhabilitada. Si se invoca este método sin primero llamar al método `stop`, o si se invoca este método cuando la JVM subyacente no da soporte a la notificación de tiempos en microsegundos, se produce una excepción `SQLException`.

getServerTimeMicros

Formato:

```
public long getServerTimeMicros()
  throws java.sql.SQLException
```

Devuelve la suma de todos los tiempos transcurridos notificados del servidor DB2 que se recogieron mientras la supervisión del sistema estaba habilitada. El tiempo está expresado en microsegundos.

El servidor DB2 notifica tiempos transcurridos cuando se dan estas condiciones:

- El servidor da soporte a la devolución de datos sobre tiempos transcurridos al cliente.
- El servidor efectúa operaciones que se pueden supervisar. Por ejemplo, el tiempo transcurrido del servidor DB2 no se devuelve para operaciones de confirmación ni retrotracción.

El tiempo transcurrido del servidor DB2 se define como el tiempo transcurrido que se invierte en analizar la corriente de datos de la petición, procesar el mandato y generar la corriente de respuesta en el servidor. El tiempo de red necesario para recibir o enviar la corriente de datos no se incluye.

Un intervalo de tiempo transcurrido de servidor DB2 es la diferencia, en microsegundos, entre estos puntos en el proceso del servidor:

Inicio de intervalo
Es el momento en el que el sistema operativo asigna a DB2 el proceso de un mensaje de TCP/IP que se ha recibido del controlador JDBC.

Fin de intervalo
Es el momento en el que DB2 está preparado para emitir el mandato de TCP/IP para devolver el mensaje de respuesta al cliente.

`getServerTimeMicros` devuelve un 0 si la supervisión del sistema está inhabilitada. Si se invoca este método sin invocar primero `stop`, se produce una excepción `SQLException`.

start

Formato:

```
public void start (int lapMode)
  throws java.sql.SQLException
```

Si el supervisor del sistema está habilitado, `start` inicia la recogida de datos de supervisión del sistema para una conexión. Los valores válidos de *lapMode* son RESET_TIMES o ACCUMULATE_TIMES.

La invocación de este método cuando la supervisión del sistema está inhabilitada no produce ningún efecto. La invocación de este método más de una vez sin que medie una llamada a `stop` produce una excepción `SQLException`.

stop

Formato:

```
public void stop()
  throws java.sql.SQLException
```

Si el supervisor del sistema está habilitado, `stop` finaliza la recogida de datos de supervisión del sistema para una conexión. Una vez detenida la supervisión, se pueden obtener los tiempos supervisados utilizando los métodos `getXXX` de `DB2SystemMonitor`.

La invocación de este método cuando la supervisión del sistema está inhabilitada no produce ningún efecto. La invocación de este método sin primero invocar `start`, o la invocación repetida de este método sin que medie una llamada a `start` produce una excepción `SQLException`.

Tareas relacionadas:

- "Conexión a una fuente de datos utilizando la interfaz DataSource" en la página 297
- "Conexión a una fuente de datos utilizando la interfaz DriverManager con el Controlador JDBC de DB2 Universal" en la página 294
- "Manejo de una excepción de SQL cuando se utiliza el Controlador JDBC de DB2 Universal" en la página 308

Información relacionada:

- "SQLCA (área de comunicaciones SQL)" en la publicación *Consulta de SQL, Volumen 1*
- "Propiedades del Controlador JDBC de DB2 Universal" en la página 400

Diferencias de JDBC entre el Controlador JDBC de DB2 Universal y otros controladores JDBC de DB2

El Controlador JDBC universal de DB2 difiere del Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows (Controlador JDBC de DB2 de tipo 2) en los aspectos siguientes:

Métodos soportados:

El Controlador JDBC universal de DB2 da soporte a diversos métodos JDBC que los demás controladores no soportan, y viceversa. Para obtener detalles, consulte el tema Comparación del soporte de controlador para las API de JDBC.

Soporte para conjuntos de resultados desplazables y actualizables:

El Controlador JDBC universal de DB2 soporta `conjuntos de resultados` desplazables y actualizables.

El Controlador JDBC de DB2 de tipo 2 soporta `conjuntos de resultados` desplazables, pero no `conjuntos de resultados` actualizables.

Diferencia en la sintaxis del URL:

La sintaxis del parámetro *url* en el método `DriverManager.getConnection` es diferente para cada controlador. Consulte los temas siguientes para obtener más información:
- Conectar a una fuente de datos utilizando la interfaz DriverManager con el Controlador JDBC de DB2 Universal
- Conectar a una fuente de datos utilizando la interfaz DriverManager con el Controlador JDBC de DB2 de tipo 2

Diferencia en los códigos de error y en los SQLSTATE devueltos para errores de controlador:

A diferencia de los demás controladores, el Controlador JDBC universal de DB2 no utiliza códigos de SQL (SQLCODE) ni estados de SQL (SQLSTATE) existentes para errores internos, . Consulte los temas Códigos de error emitidos emitidos por el Controlador JDBC de DB2 Universal y Estados de SQL emitidos por el Controlador JDBC de DB2 Universal.

El controlador JDBC/SQLJ para z/OS devuelve estados de SQL de ODBC cuando se produce un error interno.

Volumen devuelto de texto de mensajes de error:

Con el Controlador JDBC universal de DB2, cuando ejecuta `SQLException.getMessage()`, no se devuelve texto de mensajes con formato a menos que la propiedad `retrieveMessagesFromServerOnGetMessage` tenga el valor `true`.

Con el Controlador JDBC de DB2 de tipo 2, cuando ejecuta `SQLException.getMessage()`, se devuelve texto de mensajes con formato.

Mecanismos de seguridad:

Los controladores JDBC tienen diversos mecanismos de seguridad.

Para obtener información sobre los mecanismos de seguridad del Controlador JDBC universal de DB2, consulte el tema Seguridad con el Controlador JDBC de DB2 Universal.

Para obtener información sobre los mecanismos de seguridad del Controlador JDBC de DB2 de tipo 2, consulte el tema Seguridad con el Controlador JDBC de DB2 de tipo 2.

Soporte para conexiones de solo lectura:

Con el Controlador JDBC universal de DB2, puede hacer que una conexión sea de solo lectura mediante la propiedad `readOnly` de un objeto `Connection` o `DataSource`.

El Controlador JDBC de DB2 de tipo 2 utiliza el valor `Connection.setReadOnly` cuando determina si debe hacer que una conexión sea de solo lectura. Sin embargo, el definir el valor `Connection.setReadOnly(true)` no asegura que la conexión sea de solo lectura.

Resultados devueltos por ResultSet.getString para una columna BIT DATA:

El Controlador JDBC universal de DB2 devuelve datos mediante una llamada `ResultSet.getString` para una columna CHAR FOR BIT DATA o VARCHAR FOR BIT DATA en forma de serie hexadecimal en minúsculas.

El Controlador JDBC de DB2 de tipo 2 devuelve los datos en forma de serie hexadecimal en mayúsculas.

Resultado de una llamada executeUpdate que no afecta a ninguna fila:

El Controlador JDBC universal de DB2 genera un `aviso de SQL` cuando una llamada `executeUpdate` no afecta a ninguna fila.

El Controlador JDBC de DB2 de tipo 2 no devuelve un `aviso de SQL`.

Resultado de una llamada getDate o getTime para una columna TIMESTAMP:

El Controlador JDBC universal de DB2 no genera un `aviso de SQL` cuando se ejecuta una llamada `getDate` o `getTime` para una columna TIMESTAMP.

El Controlador JDBC de DB2 de tipo 2 genera un `aviso de SQL` cuando se ejecuta una llamada `getDate` o `getTime` para una columna TIMESTAMP.

Emisión de una excepción para PreparedStatement.setXXXStream con una discrepancia de longitud:

Cuando utiliza el método `PreparedStatement.setBinaryStream` , `PreparedStatement.setCharacterStream` o `PreparedStatement.setUnicodeStream`, el valor del parámetro *longitud* debe coincidir con el número de bytes de la corriente de datos de entrada.

Si los números de bytes no coinciden, el Controlador JDBC universal de DB2 no emite una excepción hasta que se ejecuta el subsiguiente método `PreparedStatement.executeUpdate`. Por tanto, para el Controlador JDBC universal de DB2, algunos datos se podrían enviar al servidor cuando las longitudes no coinciden. El servidor truncará o rellenará esos datos. La aplicación peticionaria necesita emitir una petición de retrotracción para deshacer las actualizaciones de base de datos donde están incluidos los datos truncados o rellenados.

El Controlador JDBC de DB2 de tipo 2 emite una excepción después de ejecutarse el método `PreparedStatement.setBinaryStream`, `PreparedStatement.setCharacterStream` o `PreparedStatement.setUnicodeStream`.

Correlaciones por omisión para PreparedStatement.setXXXStream:

Con el Controlador JDBC universal de DB2, cuando utiliza el método `PreparedStatement.setBinaryStream` , `PreparedStatement.setCharacterStream` o `PreparedStatement.setUnicodeStream`, y no está disponible ninguna información sobre el tipo de datos de la columna de destino, los datos de entrada se correlacionan con un tipo de datos BLOB o CLOB.

Para el Controlador JDBC de DB2 de tipo 2, los datos de entrada se correlacionan con un tipo de datos VARCHAR FOR BIT DATA o VARCHAR.

Cómo se realiza la conversión de caracteres:

Cuando se transfieren datos de tipo carácter entre un cliente y un servidor, los datos se deben convertir a un formato que el receptor pueda procesar.

Para el Controlador JDBC universal de DB2, los datos de tipo carácter que se envían desde el servidor de bases de datos al cliente se convierten utilizando los conversores internos de caracteres de Java. Las conversiones que soporta el Controlador JDBC universal de DB2 se limitan a las que soporta la implementación subyacente de JRE.

Un cliente del Controlador JDBC universal de DB2 envía datos al servidor de bases de datos en formato Unicode.

Para el controlador JDB2 de DB2 de tipo 2, las conversiones de caracteres se pueden efectuar si las conversiones están soportadas por el servidor DB2.

Conversiones de tipos de datos implícitas o explícitas para parámetros de entrada:

Si ejecuta un método `PreparedStatement.setXXX`, y el tipo de datos resultante del método `setXXX` no coincide con el tipo de datos de la columna de tabla a la que está asignado el valor de parámetro, el controlador devuelve un error a menos que se produzca una conversión del tipo de datos.

Con el Controlador JDBC universal de DB2, la conversión al tipo de datos correcto de SQL se produce implícitamente si el tipo de datos de destino es conocido y si la propiedad de conexión `deferPrepares` tiene el valor `false`. En este caso, los valores implícitos prevalecen sobre los valores explícitos que existan en la llamada de `setXXX`. Si la propiedad de conexión `deferPrepares` se establece en `true`, debe utilizar el método `PreparedStatement.setObject` para convertir el parámetro al tipo de datos correcto de SQL.

Para el Controlador JDBC de DB2 de tipo 2, si el tipo de datos de un parámetro no coincide con su tipo de datos de SQL por omisión, debe utilizar el método `PreparedStatement.setObject` para convertir el parámetro al tipo de datos correcto de SQL.

Soporte para las conversiones del tipo de datos String a BINARY para parámetros de entrada:

El Controlador JDBC universal de DB2 no da soporte a las llamadas `PreparedStatement.setObject` del formato siguiente cuando *x* es un objeto de tipo `String`:

```
setObject(parameterIndex, x, java.sqlTypes.BINARY)
```

El controlador JDBC de DB2 de tipo 2 soporta llamadas de este tipo. El controlador interpreta el valor de *x* como una serie de caracteres hexadecimal.

Resultado de PreparedStatement.setObject cuando existe una discrepancia de escala decimal:

Con el Controlador JDBC universal de DB2, si invoca `PreparedStatement.setObject` con un parámetro de entrada decimal y la escala del parámetro de entrada es mayor que la escala de la columna de destino, el controlador trunca los dígitos de cola del valor de entrada antes de asignar el valor a la columna.

El Controlador JDBC de DB2 de tipo 2 redondea los dígitos de cola del valor de entrada antes de asignar el valor a la columna.

Soporte para conversiones desde el tipo de datos java.lang.Character para parámetros de entrada:

Para el formato siguiente de `PreparedStatement.setObject`, el the Controlador JDBC universal de DB2 soporta las correlaciones de tipos de datos estándar de objetos Java a tipos de datos JDBC cuando convierte *x* a un tipo de datos JDBC:
`setObject(parameterIndex, x)`

El Controlador JDBC de DB2 de tipo 2 soporta la correlación no estándar de *x* desde `java.lang.Character` a CHAR.

Soporte para ResultSet.getBinaryStream con una columna de caracteres:

El Controlador JDBC universal de DB2 soporta `ResultSet.getBinaryStream` con un argumento que representa una columna de caracteres solo si la columna tiene el atributo FOR BIT DATA.

Para el Controlador JDBC de DB2 de tipo 2, si el argumento `ResultSet.getBinaryStream` es una columna de caracteres, esa columna no necesita tener el atributo FOR BIT DATA.

Datos devueltos por ResultSet.getBinaryStream para una columna binaria:

Con el Controlador JDBC universal de DB2, cuando ejecuta `ResultSet.getBinaryStream` para una columna binaria, los datos devueltos están en minúsculas, en forma de pares de dígitos hexadecimales.

Con el Controlador JDBC de DB2 de tipo 2, cuando ejecuta `ResultSet.getBinaryStream` para una columna binaria, los datos devueltos están en mayúsculas, en forma de pares de dígitos hexadecimales.

Resultado de utilizar setObject con Boolean como tipo de datos de entrada y CHAR como tipo de datos de destino:

Con el Controlador JDBC universal de DB2, cuando ejecuta `PreparedStatement.setObject(parameterIndex,x,CHAR)`, y *x* es de tipo Boolean, se inserta el valor ″0″ o ″1″ en la columna de tabla.

Con el Controlador JDBC de DB2 de tipo 2, se inserta la serie ″false″ o ″true″ en la columna de tabla. La longitud de la columna de tabla debe ser 5 como mínimo.

Resultado de utilizar getBoolean para recuperar un valor de una columna CHAR:

Con el Controlador JDBC universal de DB2, cuando ejecuta `ResultSet.getBoolean` o `CallableStatement.getBoolean` para recuperar un valor booleano de una columna CHAR, y la columna contiene el valor ″false″ o ″0″, se devuelve el valor `false`. Si la columna contiene cualquier otro valor, se devuelve `true`.

Con el Controlador JDBC de DB2 de tipo 2, cuando ejecuta `ResultSet.getBoolean` o `CallableStatement.getBoolean` para recuperar un valor booleano de una columna CHAR, y la columna contiene el valor ″true″ o ″1″, se devuelve el valor `true`. Si la columna contiene cualquier otro valor, se devuelve `false`.

Resultado de ejecutar ResultSet.next() para un cursor cerrado:

Con el Controlador JDBC universal de DB2, cuando ejecuta `ResultSet.next()` para un cursor cerrado, se emite una excepción `SQLException`. Esto se ajusta al estándar JDBC.

Con el Controlador JDBC de DB2 de tipo 2, cuando ejecuta `ResultSet.next()` para un cursor cerrado, se devuelve el valor `false` y no se emite ninguna excepción.

Resultado de especificar argumentos nulos en llamadas DatabaseMetaData:

Con el Controlador JDBC universal de DB2, puede especificar `null` como argumento de una llamada `DatabaseMetaData` solo donde la especificación JDBC especifique que `null` está permitido. De lo contrario, se emite un excepción.

Con el Controlador JDBC de DB2 de tipo 2, el valor `null` significa que el argumento no se utiliza para restringir la búsqueda.

Soporte para el tipo DATALINK:

El Controlador JDBC universal de DB2 no da soporte al tipo DATALINK de SQL.

El Controlador JDBC de DB2 de tipo 2 soporta el tipo DATALINK en las llamadas de método con estos formatos:
- `PreparedStatement.setObject(`*`parameterIndex`*`, `*`x`*`, DB2Constants.DATALINK)`
- `PreparedStatement.setObject(`*`parameterIndex`*`, `*`x`*`, java.sql.Types.DATALINK)` (Java 1.4 o posterior)
- `PreparedStatement.setURL(` *`parameterIndex`*`, java.net.URL)`
- `PreparedStatement.setObject(` *`parameterIndex`*`, java.net.URL)`
- `PreparedStatement.setObject(`*`parameterIndex`*`, java.net.URL, java.sql.Types.DATALINK)` (Java 1.4 o posterior)
- `ResultSet.getString` para una columna DATALINK
- `ResultSet.getURL` para una columna DATALINK

Conversión a mayúsculas de argumentos de método:

El Controlador JDBC universal de DB2 no convierte a mayúsculas los argumentos de las llamadas de método.

El Controlador JDBC de DB2 de tipo 2 convierte a mayúsculas el argumento de una llamada `Statement.setCursorName`. Para impedir que el nombre de cursor se convierta a mayúsculas, encierre el nombre de cursor entre caracteres \". Por ejemplo:

```
Statement.setCursorName("\"micursor\"");
```

Soporte para cláusulas de escape de indicación horaria:

El Controlador JDBC universal de DB2 soporta el formato estándar de una cláusula de escape de `TIME`:

```
{t 'hh:mm:ss'}
```

Además del formato estándar, el Controlador JDBC de DB2 de tipo 2 soporta el formato siguiente de una cláusula de escape de `TIME`:

```
{ts 'hh:mm:ss'}
```

Uso de literales o expresiones como parámetros de la sentencia CALL:

El Controlador JDBC universal de DB2 soporta el uso de solamente marcadores de parámetros como parámetros de la sentencia CALL.

El Controlador JDBC de DB2 de tipo 2 soporta el uso de literales y marcadores de parámetros como parámetros de la sentencia CALL.

Inclusión de una sentencia CALL en un lote de sentencias:

El Controlador JDBC universal de DB2 soporta el uso de sentencias CALL en un lote de sentencias.

El Controlador UDB de DB2 de tipo 2 no soporta el uso de sentencias CALL en un lote de sentencias.

Eliminación de caracteres sobrantes en el texto de una sentencia de SQL:

El Controlador JDBC universal de DB2 no elimina los caracteres de espacio en blanco, tales como espacios, tabuladores y caracteres de línea nueva, en el texto de sentencias de SQL antes de pasar ese texto al servidor de bases de datos.

El Controlador UDB de DB2 de tipo 2 elimina los caracteres de espacio en blanco en el texto de la sentencia de SQL antes de pasar ese texto al servidor de bases de datos.

Resultado de ejecutar PreparedStatement.executeBatch:

Con el Controlador JDBC universal de DB2, cuando se ejecuta una sentencia PreparedStatement.executeBatch, el controlador devuelve una matriz `int` de cuentas de actualización. Cada elemento de la matriz contiene el número de filas que fueron actualizadas por una sentencia del lote de sentencias.

Con el Controlador UDB de DB2 de tipo 2, cuando se ejecuta una sentencia PreparedStatement.executeBatch, el controlador no puede determinar las cuentas de actualización, por lo que devuelve el valor -3 para cada cuenta de actualización.

Soporte para SQL compuesto:

El Controlador JDBC universal de DB2 no da soporte al uso de bloques de SQL compuesto.

El Controlador UDB de DB2 de tipo 2 soporta el uso de bloques de SQL compuesto. Puede utilizar una llamada `PreparedStatement.executeUpdate` o `Statement.executeUpdate` para ejecutar un bloque de SQL compuesto.

Resultado de no definir un parámetro en una actualización por lotes:

El Controlador JDBC universal de DB2 emite una excepción después de una llamada `PreparedStatement.addBatch` si un parámetro no está definido.

El Controlador UDB de DB2 de tipo 2 emite una excepción después de una llamada `PreparedStatement.executeBatch` si un parámetro no está definido para cualquiera de las sentencias del lote.

Capacidad para invocar procedimientos almacenados no catálogados:

El Controlador JDBC universal de DB2 no permite invocar procedimientos almacenados que no estén definidos en el catálogo de DB2.

El Controlador UDB de DB2 de tipo 2 permite invocar procedimientos almacenados que no estén definidos en el catálogo de DB2.

Conceptos relacionados:

- "Seguridad cuando se utiliza el Controlador JDBC de DB2 Universal" en la página 478
- "Datos LOB en aplicaciones JDBC con el Controlador JDBC de DB2 Universal" en la página 315
- "Seguridad cuando se utiliza el Controlador JDBC de DB2 de tipo 2" en la página 477

Tareas relacionadas:

- "Realización de actualizaciones por lotes en aplicaciones JDBC" en la página 330
- "Utilización de métodos CallableStatement para llamar a procedimientos almacenados" en la página 306
- "Conexión a una fuente de datos utilizando la interfaz DataSource" en la página 297
- "Conexión a una fuente de datos utilizando la interfaz DriverManager con el Controlador JDBC de DB2 Universal" en la página 294
- "Manejo de una excepción de SQL cuando se utiliza el Controlador JDBC de DB2 Universal" en la página 308
- "Uso del método PreparedStatement.executeUpdate para actualizar datos de tablas DB2" en la página 303
- "Especificación de las capacidades de actualización, desplazamiento y retención para conjuntos de resultados en aplicaciones de JDBC" en la página 335
- "Utilización del método Statement.executeUpdate para crear y modificar objetos DB2" en la página 301

Información relacionada:

- "Propiedades del Controlador JDBC de DB2 Universal" en la página 400
- "Códigos de error emitidos por el Controlador JDBC de DB2 Universal" en la página 468
- "Estados de SQL emitidos por el Controlador JDBC de DB2 Universal" en la página 469
- "Comparación del soporte de controlador para las API de JDBC" en la página 407
- "Tipos de datos de Java, JDBC y SQL" en la página 395

Diferencias respecto a SQLJ entre el Controlador JDBC universal de DB2 y otros controladores JDBC de DB2

El soporte de SQLJ en el Controlador JDBC universal de DB2 difiere en los aspectos siguientes del soporte de SQLJ existente en los demás controladores JDBC de DB2:

Diferencias en los perfiles serializados:

El Controlador JDBC de DB2 de tipo 2 y el Controlador JDBC universal de DB2 crean código binario diferente cuando se ejecutan sus programas de utilidad de conversión y personalización de SQLJ. Por tanto, las aplicaciones SQLJ que convirtió y personalizó utilizando los programas de utilidad `sqlj` y `db2profc` del Controlador JDBC de DB2 de tipo 2 no se ejecutarán con el Controlador JDBC universal de DB2. ***Para poder ejecutar esas aplicaciones SQLJ con el Controlador JDBC universal de DB2, las tiene que volver a convertir y personalizar utilizando los programas de utilidad `sqlj` y `db2sqljcustomize` del Controlador JDBC universal de DB2.*** Debe hacer esto aunque no hubiera modificado las aplicaciones.

Soporte de la sentencia VALUES de SQL:

El Controlador JDBC de DB2 de tipo 2 soporta la sentencia VALUES de SQL en una cláusula de una sentencia de SQLJ, pero no así el Controlador JDBC universal de DB2. Por tanto, es necesario que modifique las aplicaciones SQLJ que incluyan sentencias VALUES.

Ejemplo: suponga que un programa SQLJ contiene la sentencia siguiente:

```
#sql [ctxt] hv = {VALUES (MY_ROUTINE(1))};
```

Para el Controlador JDBC universal de DB2, es necesario que cambie esa sentencia por lo siguiente:

```
#sql [ctxt] {SELECT MY_ROUTINE(1) INTO :hv FROM SYSIBM.SYSDUMMY1};
```

Soporte de sentencias de SQL compuesto:

El Controlador JDBC de DB2 de tipo 2 soporta sentencias de SQL compuesto en una cláusula de una sentencia de SQLJ, pero no así el Controlador JDBC universal de DB2. Por tanto, es necesario que modifique las aplicaciones SQLJ que tengan sentencias de SQLJ donde se incluyan las especificaciones BEGIN COMPOUND y END COMPOUND. Si utiliza sentencias compuestas para efectuar actualizaciones de proceso por lotes, en su lugar puede utilizar las interfaces de programación de SQLJ para las actualizaciones de proceso por lotes.

Diferencia en las técnicas de conexión:

Las técnicas de conexión que están disponibles, y los nombres de controlador y los URL que se utilizan para esas técnicas de conexión, varían de un controlador a otro. Consulte el tema Conectar a una fuente de datos utilizando SQLJ para obtener más información.

Soporte para iteradores desplazables y actualizables:

SQLJ con el Controlador JDBC universal de DB2 soporta iteradores desplazables y actualizables.

El Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows (Controlador JDBC de DB2 de tipo 2) soporta cursores desplazables, pero no iteradores actualizables.

Ejecución dinámica de sentencias de SQL con WebSphere Application Server:

Con el Controlador JDBC universal de DB2, si está utilizando una versión de WebSphere Application Server anterior a la versión 5.0.1, todas las sentencias de SQL de un programa SQLJ se ejecutan dinámicamente, con independencia de si el

usuario personaliza el programa SQLJ. Para WebSphere Application Server Versión 5.0.1 y versiones posteriores, si personaliza el programa SQLJ, las sentencias de SQL se ejecutan de forma estática.

Conceptos relacionados:

- "Seguridad cuando se utiliza el Controlador JDBC de DB2 Universal" en la página 478
- "Datos LOB en aplicaciones JDBC con el Controlador JDBC de DB2 Universal" en la página 315
- "Seguridad cuando se utiliza el Controlador JDBC de DB2 de tipo 2" en la página 477

Tareas relacionadas:

- "Realización de actualizaciones por lotes en aplicaciones JDBC" en la página 330
- "Utilización de métodos CallableStatement para llamar a procedimientos almacenados" en la página 306
- "Conexión a una fuente de datos utilizando la interfaz DataSource" en la página 297
- "Conexión a una fuente de datos utilizando la interfaz DriverManager con el Controlador JDBC de DB2 Universal" en la página 294
- "Manejo de una excepción de SQL cuando se utiliza el Controlador JDBC de DB2 Universal" en la página 308
- "Uso del método PreparedStatement.executeUpdate para actualizar datos de tablas DB2" en la página 303
- "Especificación de las capacidades de actualización, desplazamiento y retención para conjuntos de resultados en aplicaciones de JDBC" en la página 335
- "Utilización del método Statement.executeUpdate para crear y modificar objetos DB2" en la página 301

Información relacionada:

- "Propiedades del Controlador JDBC de DB2 Universal" en la página 400
- "Códigos de error emitidos por el Controlador JDBC de DB2 Universal" en la página 468
- "Estados de SQL emitidos por el Controlador JDBC de DB2 Universal" en la página 469
- "Comparación del soporte de controlador para las API de JDBC" en la página 407
- "Tipos de datos de Java, JDBC y SQL" en la página 395

Códigos de error emitidos por el Controlador JDBC de DB2 Universal

Los códigos de error comprendidos dentro de los rangos +4200 a +4299, +4450 a +4499, -4200 a -4299, y -4450 a -4499 están reservados para el Controlador JDBC universal de DB2. Actualmente, el Controlador JDBC universal de DB2 emite los códigos de error siguientes:

-4200 Una aplicación perteneciente a una transacción global en un entorno XA ha emitido una operación de confirmación o retrotracción no válida.

-4498 Se ha producido una gestión de anomalías o recuperación y la transacción ha fallado.

-4499 Se ha producido un error grave que ha dado como resultado una desconexión.

-99999 El Controlador JDBC universal de DB2 ha emitido un error que no tiene asignado todavía un código de error.

Tareas relacionadas:

- "Manejo de una excepción de SQL cuando se utiliza el Controlador JDBC de DB2 Universal" en la página 308
- "Manejo de errores de SQL en una aplicación SQLJ" en la página 371

Información relacionada:

- "Diferencias de JDBC entre el Controlador JDBC de DB2 Universal y otros controladores JDBC de DB2" en la página 459

Estados de SQL emitidos por el Controlador JDBC de DB2 Universal

Los estados de SQL (SQLSTATE) comprendidos dentro del rango 46600 a 466ZZ están reservados para el Controlador JDBC universal de DB2. Actualmente el Controlador JDBC universal de DB2 devuelve un valor de SQLSTATE nulo para un error interno, a menos que el error sea un error de DRDA. Se emiten los SQLSTATE siguientes para errores de DRDA:

08004 El servidor de aplicaciones ha rechazado el establecimiento de la conexión.

22021 Un carácter no se encuentra en el conjunto de caracteres codificados.

24501 El cursor identificado no está abierto.

2D521 Las operaciones COMMIT o ROLLBACK de SQL no son válidas en el entorno operativo actual.

58008 La ejecución ha fallado debido a un error del protocolo de distribución que no afectará a la ejecución satisfactoria de los subsiguientes mandatos de DDM o sentencias de SQL.

58009 La ejecución ha fallado debido a un error del protocolo de distribución que ha provocado la desasignación de la conversación.

58010 La ejecución ha fallado debido a un error del protocolo de distribución que impedirá que los subsiguientes mandatos de DDM o sentencias de SQL se ejecuten satisfactoriamente.

58014 No se da soporte al mandato de DDM.

58015 No se da soporte al objeto de DDM.

58016 No se da soporte al parámetro de DDM.

58017 No se da soporte al valor del parámetro de DDM.

Tareas relacionadas:

- "Manejo de una excepción de SQL cuando se utiliza el Controlador JDBC de DB2 Universal" en la página 308
- "Manejo de errores de SQL en una aplicación SQLJ" en la página 371

Información relacionada:

- "Diferencias de JDBC entre el Controlador JDBC de DB2 Universal y otros controladores JDBC de DB2" en la página 459

Capítulo 18. Instalación de los controladores JDBC

Las secciones siguientes contienen información sobre la instalación de los controladores JDBC.

Instalación del Controlador JDBC de DB2 Universal

Si selecciona el soporte de JDBC durante la instalación de cualquiera de los productos DB2 UDB para Linux, UNIX y Windows, el programa de instalación ejecuta algunos de los pasos de instalación para el Controlador JDBC universal de DB2. El programa de instalación de DB2 UDB Versión 8 para Windows realiza las tareas siguientes:

- Instala los archivos db2jcc.jar y sqlj.zip y los añade a la variable CLASSPATH del sistema
- Instala el archivo db2jcct2.dll, que es necesario para Conectividad de tipo 2 universal, en el directorio sqllib\bin

El programa de instalación de DB2 UDB Versión 8 para UNIX realiza las tareas siguientes:

- Instala los archivos db2jcc.jar y sqlj.zip
- Añade los archivos db2jcc.jar y sqlj.zip a la sentencia CLASSPATH del script db2profile (para el shell Bourne o Korn) o del script db2cshrc (para el shell C)
- Instala el archivo libdb2jcct2.so (libdb2jcct2.sl para HP-UX), que es necesario para Conectividad de tipo 2 universal, en el directorio sqllib/lib

Debe seguir los pasos adicionales siguientes para instalar el Controlador JDBC universal de DB2.

Requisitos necesarios:

- JDK 1.3.1 o posterior

  Las funciones de JDBC 3.0 necesitan un nivel mínimo de JDK igual a 1.4.
- TCP/IP

  Los servidores se deben configurar para la comunicación TCP/IP en estos casos:
  - Las aplicaciones JDBC o SQLJ utilizan Conectividad de tipo 4 universal.
  - Las aplicaciones JDBC o SQLJ utilizan Conectividad de tipo 2 universal, y especifican el *servidor* y el *puerto* en el URL de conexión.
- Procedimientos almacenados de DB2 UDB para z/OS u OS/390

  Si cualquiera de las aplicaciones JDBC o SQLJ se conectará a un servidor de DB2 UDB para z/OS u OS/390, es necesario instalar varios procedimientos almacenados en ese servidor para permitir la recuperación de información del catálogo DB2, la función de rastreo y el formateo de los mensajes de error. Los procedimientos almacenados son:
  - SQLCOLPRIVILEGES
  - SQLCOLUMNS
  - SQLFOREIGNKEYS
  - SQLGETTYPEINFO
  - SQLPRIMARYKEYS
  - SQLPROCEDURECOLS
  - SQLPROCEDURES
  - SQLSPECIALCOLUMNS
  - SQLSTATISTICS

– SQLTABLEPRIVILEGES
– SQLTABLES
– SQLUDTS
– SQLCAMESSAGE

Los procedimientos almacenados se proporcionan con el producto DB2 UDB para z/OS Versión 8. Los procedimientos almacenados se proporcionan en los PTF siguientes de DB2 UDB para OS/390 y z/OS Versión 7 o de DB2 UDB para OS/390 Versión 6:

Tabla 75. Lista de los PTF de DB2 UDB para OS/390 y z/OS

Versión de DB2 para OS/390 y z/OS	Número de PTF
Versión 6	UQ72081 y UQ72082
Versión 7	UQ72083

Consulte al administrador del sistema DB2 UDB para z/OS para conocer si estos procedimientos almacenados están instalados.

- Soporte de Unicode para servidores iSeries

  Si cualquiera de los programas SQLJ o JDBC utilizará Conectividad de tipo 4 universal para conectar con un servidor DB2 UDB para iSeries, el sistema operativo OS/400 debe dar soporte al esquema de codificación UTF-8 de Unicode. La tabla siguiente lista los PTF de OS/400 que son necesarios para dar soporte a UTF-8 de Unicode:

Tabla 76. Lista de los PTF de OS/400 para el soporte UTF-8 de Unicode

Versión de OS/400	Número de PTF
V5R3 o posterior	Ninguno (el soporte está incluido)
V5R2	SI06541, SI06796, SI07557, SI07564, SI07565, SI07566 y SI07567
V5R1	SI06308, SI06300, SI06301, SI06302, SI06305, SI06307 y SI05872

- Soporte Java para clientes y servidores HP-UX

  Servidores HP-UX: El Controlador JDBC universal de DB2 no da soporte a las bases de datos que hacen uso del juego de caracteres por omisión de HP-UX, Roman8. Por tanto, cuando cree una base de datos en un servidor HP-UX al que piense acceder mediante el Controlador JDBC universal de DB2, es necesario que cree la base de datos con un juego de caracteres diferente.

  Clientes y servidores HP-UX: en un sistema HP-UX, el entorno Java necesita aplicaciones especiales de configuración en el Controlador JDBC universal de DB2.

  Consulte los temas Configuración del entorno Java para HP-UX para conocer detalles.

Procedimiento:

Nota:

Para instalar el Controlador JDBC universal de DB2:

1. Incluya el archivo db2jcc.jar en la variable CLASSPATH de la aplicación Java, de forma que preceda a cualquier otro archivo JAR de otros controladores JDBC.
2. Si piensa utilizar la seguridad Kerberos, coloque los archivos siguientes en la variable CLASSPATH de la aplicación Java:

- ibmjceprovider.jar
- ibmjcefw.jar
- ibmjlog.jar
- US_export_policy.jar
- Local_policy.jar
- ibmjgssprovider.jar
- jaas.jar
- ibmjceprovider.jar
- ibmjcefw.jar
- ibmjlog.jar
- US_export_policy.jar
- Local_policy.jar

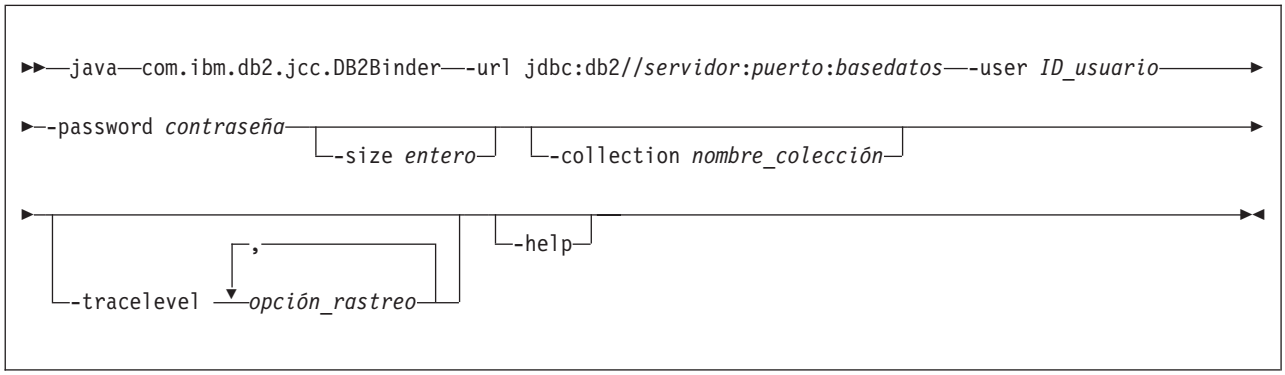
3. Incluya uno o más nombres de archivos de licencia en la variable CLASSPATH de la aplicación Java para permitir la conectividad a servidores. La Tabla 77 lista esos archivos de licencia.

Tabla 77. Archivos de licencia del Controlador JDBC de DB2 Universal

Archivo de licencia	Servidor con el cual el archivo de licencia permite conectar	Producto donde se incluye el archivo de licencia
db2jcc_license_c.jar	Cloudscape	Cloudscape Network Server
db2jcc_license_cu.jar	Cloudscape DB2 UDB para Linux, UNIX y Windows	DB2 UDB para Linux, UNIX y Windows
db2jcc_license_cisuz.jar	Cloudscape DB2 UDB para Linux, UNIX y Windows DB2 UDB para z/OS o DB2 UDB para OS/390 y z/OS DB2 UDB para iSeries DB2 Server para VSE y VM	DB2 Connect

4. Si piensa utilizar SQLJ, añada sqlj.zip a la variable CLASSPATH de la aplicación Java. Elimine las versiones antiguas de sqlj.zip y runtime.zip en la variable CLASSPATH.
5. Si piensa conectar con un servidor DB2 UDB para z/OS u OS/390, ejecute el programa de utilidad com.ibm.db2.jcc.DB2Binder para vincular los paquetes DB2 que son utilizados por el Controlador JDBC universal de DB2 en el servidor.

   Sintaxis de DB2Binder:



Descripciones de las opciones de DB2Binder:

-url
Especifica la fuente de datos en donde se deben vincular los paquetes JCC. Las partes variables del valor `-url` son:

servidor
El nombre de dominio o dirección IP del sistema MVS donde reside el subsistema DB2.

puerto
El número de puerto del servidor TCP/IP que está asignado al subsistema DB2. El valor por omisión es 446.

basedatos
El nombre de ubicación del subsistema DB2, tal como está definido en la tabla de catálogo SYSIBM.LOCATIONS.

-user
Especifica el ID de usuario utilizado para vincular los paquetes. Este usuario debe tener autorización BIND sobre los paquetes.

-size
Especifica el número de paquetes DB2 que DB2binder vincula para cada uno de los cuatro niveles de aislamiento de DB2 y cada uno de los dos valores de holdability. El Controlador JDBC universal de DB2 utiliza estos paquetes para procesar SQL dinámico. Además, el programa DB2binder vincula un paquete que el Controlador JDBC universal de DB2 utiliza para el SQL estático. Por tanto, el número total de paquetes que DB2binder vincula es:

`4*2*`*`entero`*`+1`

El valor por omisión de *entero* es 3.

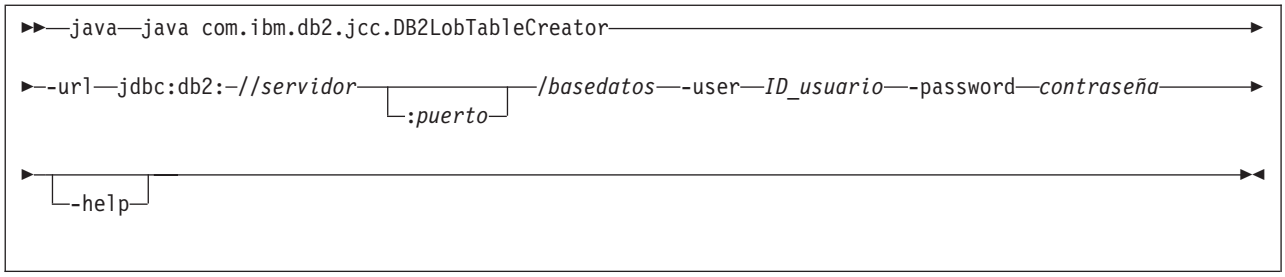
-collection
Especifica el ID de colección de los paquetes que son utilizados por una instancia del Controlador JDBC universal de DB2. El valor por omisión es NULLID. DB2binder convierte este valor a mayúsculas.

Puede crear varias instancias del conjunto de paquetes JCC en una sola ubicación ejecutando varias veces `com.ibm.db2.jcc.DB2Binder` y especificando un valor diferente para -collection cada vez. Durante la ejecución, seleccione una instancia del Controlador JDBC universal de DB2 asignando a la propiedad `currentPackageSet` un valor que coincida con valor de -collection. Consulte Propiedades del Controlador JDBC de DB2 Universal para obtener información sobre la propiedad `currentPackageSet`.

-tracelevel
Especifica qué se debe rastrear mientras se ejecuta DB2Binder. Consulte la explicación de la propiedad `traceLevel` en el tema Propiedades del Controlador JDBC de DB2 Universal para conocer las opciones disponibles.

6. Si piensa utilizar localizadores de LOB para acceder a columnas DBCLOB de tablas DB2 en un servidor DB2 UDB para z/OS u OS/390, ejecute el programa de utilidad com.ibm.db2.jcc.DB2LobTableCreator para cada uno de esos servidores para crear las tablas necesarias para obtener localizadores de LOB.

   Sintaxis de DB2LobTableCreator:



Descripciones de las opciones de DB2LobTableCreator:

-url
Especifica la fuente de datos en donde se debe ejecutar DB2LobTableCreator. Las partes variables del valor `-url` son:

jdbc:db2:
Indica que la conexión es con un servidor perteneciente a la familia de productos DB2 UDB.

servidor
El nombre de dominio o dirección IP del servidor de bases de datos.

puerto
El número de puerto del servidor TCP/IP que está asignado al servidor de bases de datos. Es un valor entero comprendido entre 0 y 65535. El valor por omisión es 446.

basedatos
Nombre del servidor de bases de datos.

basedatos es el nombre de ubicación DB2 que se define durante la instalación. Todos los caracteres de este valor deben ser caracteres en mayúsculas. Puede determinar el nombre de ubicación ejecutando la sentencia de SQL siguiente en el servidor:

```
SELECT CURRENT SERVER FROM SYSIBM.SYSDUMMY1;
```

-user
Especifica el ID de usuario utilizado para ejecutar DB2LobTableCreator. Este usuario debe tener autorización para crear tablas en la base de datos DSNATPDB.

-password
Especifica la contraseña del ID de usuario.

-help
Especifica que el programa de utilidad DB2LobTableCreator describe todas las opciones a las que da soporte. Si se especifica cualquier otra opción con -help, no se tiene en cuenta.

Tareas relacionadas:

- "Configuración del entorno Java para HP-UX" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Configuración de comunicaciones de TCP/IP para una instancia DB2" en la publicación *Suplemento de instalación y configuración*
- "Actualización del archivo de configuración del gestor de bases de datos en el servidor para las comunicaciones de TCP/IP" en la publicación *Suplemento de instalación y configuración*
- "Actualización del archivo de servicios en el servidor para las comunicaciones de TCP/IP" en la publicación *Suplemento de instalación y configuración*

Información relacionada:

- "Propiedades del Controlador JDBC de DB2 Universal" en la página 400

Capítulo 19. Seguridad en JDBC y SQLJ

Las secciones siguientes contienen información sobre los mecanismos de seguridad que están disponibles para los controladores JDBC.

Seguridad cuando se utiliza el Controlador JDBC de DB2 de tipo 2

El Controlador JDBC de DB2® de tipo 2 para Linux, UNIX® y Windows® (Controlador JDBC de DB2 de tipo 2) soporta el uso de un mecanismo de seguridad basado en un ID de usuario y una contraseña. Debe establecer el ID de usuario y la contraseña o no definir ninguno de ellos. Si no define un ID de usuario y contraseña, el controlador utiliza el ID de usuario y contraseña del usuario que está conectado actualmente al sistema operativo.

Para especificar la seguridad basada en un ID de usuario y una contraseña para una conexión JDBC, utilice una de las técnicas siguientes.

Para la interfaz `DriverManager`: puede especificar el ID de usuario y la contraseña directamente en la invocación de `DriverManager.getConnection`. Por ejemplo:

```
import java.sql.*;      // Base JDBC
...
String id = "db2adm";                // Definir ID de usuario
Sring pw = "db2adm";                 // Definir contraseña
String url = "jdbc:db2:toronto";
                           // Definir URL para fuente de datos
Connection con = DriverManager.getConnection(url, id, pw);
                          // Crear conexión
```

Como alternativa, puede establecer el ID de usuario y la contraseña definiendo las propiedades `user` y `password` en un objeto `Properties`, y luego invocar la modalidad del método `getConnection` que hace uso del objeto `Properties` como parámetro. Por ejemplo:

```
import java.sql.*;      // Base JDBC
import COM.ibm.db2.jdbc.*;  // Implementación de JDBC 2.0 para DB2
...
Properties properties = new java.util.Properties();
                                    // Crear objeto Properties
properties.put("user", "db2adm");    // Definir ID de usuario para conexión
properties.put("password", "db2adm");  // Definir contraseña para conexión
String url = "jdbc:db2:toronto";
                                     // Definir URL para fuente de datos
Connection con = DriverManager.getConnection(url, properties);
                                    // Crear conexión
```

Para la interfaz `DataSource`: puede especificar el ID de usuario y la contraseña directamente en la invocación de `DataSource.getConnection`. Por ejemplo:

```
import java.sql.*;      // Base JDBC
import COM.ibm.db2.jdbc.*;  // Implementación de JDBC 2.0 para DB2
...
Context ctx=new InitialContext();     // Crear contexto para JNDI
DataSource ds=(DataSource)ctx.lookup("jdbc/sampledb");
                                    // Obtener objeto DataSource
String id = "db2adm";                // Definir ID de usuario
Sring pw = "db2adm";                 // Definir contraseña
Connection con = ds.getConnection(id, pw);
                                   // Crear conexión
```

Como alternativa, si crea y despliega el objeto DataSource, puede establecer el ID de usuario y la contraseña invocando los métodos DataSource.setUser y DataSource.setPassword después de crear el objeto DataSource. Por ejemplo:

```
import java.sql.*;      // Base JDBC
import COM.ibm.db2.jdbc.*;   // Implementación de JDBC 2.0 para DB2
...
DB2DataSource db2ds = new DB2DataSource();
                                  // Crear el objeto DataSource
db2ds.setDatabaseName("toronto");   // Definir la ubicación
db2ds.setUser("db2adm");           // Definir el ID de usuario
db2ds.setPassword("db2adm");       // Definir la contraseña
```

Conceptos relacionados:

- "Conexión de aplicaciones DB2 a una fuente de datos utilizando la interfaz DriverManager con el Controlador JDBC de DB2 de tipo 2" en la página 292

Tareas relacionadas:

- "Conexión a una fuente de datos utilizando la interfaz DataSource" en la página 297
- "Creación y despliegue de objetos DataSource" en la página 338

Seguridad cuando se utiliza el Controlador JDBC de DB2 Universal

Cuando utiliza el Controlador JDBC universal de DB2, puede seleccionar un mecanismo de seguridad especificando un valor para la propiedad securityMechanism. Puede definir esta propiedad en una de las maneras siguientes:

- Si utiliza la interfaz DriverManager, defina securityMechanism en un objeto java.util.Properties antes de invocar la modalidad del método getConnection que hace uso del parámetro java.util.Properties.
- Si utiliza la interfaz DataSource, y está creando y desplegando sus propios objetos DataSource, invoque el método DataSource.setSecurityMechanism después de crear un objeto DataSource.

La Tabla 78 muestra los mecanismos de seguridad soportados por el Controlador JDBC universal de DB2, y el valor debe especificar para la propiedad securityMechanism al seleccionar un mecanismo de seguridad. El mecanismo de seguridad por omisión es el basado en el ID de usuario y la contraseña.

Tabla 78. Mecanismos de seguridad soportados por el Controlador JDBC universal de DB2

Mecanismo de seguridad	Valor de la propiedad securityMechanism
ID de usuario y contraseña	DB2BaseDataSource.CLEAR_TEXT_PASSWORD_SECURITY
ID de usuario solamente	DB2BaseDataSource.USER_ONLY_SECURITY
ID de usuario y contraseña cifrada	DB2BaseDataSource.ENCRYPTED_PASSWORD_SECURITY
ID de usuario cifrado y contraseña cifrada	DB2BaseDataSource.ENCRYPTED_USER_AND_PASSWORD_SECURITY
Kerberos[1]	DB2BaseDataSource.KERBEROS_SECURITY

Nota:

1. Disponible solo para Conectividad de tipo 4 universal.

Conceptos relacionados:

- "Seguridad mediante ID de usuario cifrado o contraseña cifrada con el Controlador JDBC de DB2 Universal" en la página 481

- "Seguridad Kerberos con el Controlador JDBC de DB2 Universal" en la página 482
- "Seguridad basada solo en el ID de usuario cuando se utiliza el Controlador JDBC de DB2 Universal" en la página 480
- "Seguridad basada en ID de usuario y contraseña cuando se utiliza el Controlador JDBC de DB2 Universal" en la página 479

Información relacionada:

- "Propiedades del Controlador JDBC de DB2 Universal" en la página 400

Seguridad basada en ID de usuario y contraseña cuando se utiliza el Controlador JDBC de DB2 Universal

Para especificar la seguridad basada en un ID de usuario y una contraseña para una conexión JDBC, utilice una de las técnicas siguientes.

Para la interfaz `DriverManager`: puede especificar el ID de usuario y la contraseña directamente en la invocación de `DriverManager.getConnection`. Por ejemplo:

```
import java.sql.*;      // Base JDBC
...
String id = "db2adm";      // Definir ID de usuario
String pw = "db2adm";      // Definir contraseña
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                           // Definir URL para fuente de datos
Connection con = DriverManager.getConnection(url, id, pw);
                           // Crear conexión
```

Otro método consiste en definir el ID de usuario y la contraseña directamente en la serie de caracteres del URL. Por ejemplo:

```
import java.sql.*;      // Base JDBC
...
String url =
  "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose:user=db2adm;password=db2adm;";
                           // Definir URL para fuente de datos
Connection con = DriverManager.getConnection(url);
                           // Crear conexión
```

Como alternativa, puede establecer el ID de usuario y la contraseña definiendo las propiedades `user` y `password` en un objeto `Properties`, y luego invocar la modalidad del método `getConnection` que hace uso del objeto `Properties` como parámetro. Opcionalmente, puede definir la propiedad `securityMechanism` para indicar que está utilizando la seguridad basada en un ID de usuario y contraseña. Por ejemplo:

```
import java.sql.*;      // Base JDBC
import com.ibm.db2.jcc.*;     // Implementación de JDBC 2.0 para DB2®
...
Properties properties = new java.util.Properties();
                                    // Crear objeto Properties
properties.put("user", "db2adm");  // Definir ID de usuario para conexión
properties.put("password", "db2adm");  // Definir contraseña para conexión
properties.put("securityMechanism",
  new String("" + com.ibm.db2.jcc.DB2BaseDataSource.CLEAR_TEXT_PASSWORD_SECURITY +
  ""));
                             // Definir el mecanismo de seguridad como
                             // ID de usuario y contraseña
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
```

```
                            // Definir URL para fuente de datos
Connection con = DriverManager.getConnection(url, properties);
                            // Crear conexión
```

Para la interfaz DataSource: puede especificar el ID de usuario y la contraseña directamente en la invocación de `DataSource.getConnection`. Por ejemplo:

```
import java.sql.*;      // Base JDBC
import com.ibm.db2.jcc.*;     // Implementación de JDBC 2.0 para DB2
...
Context ctx=new InitialContext();     // Crear contexto para JNDI
DataSource ds=(DataSource)ctx.lookup("jdbc/sampledb");
                                      // Obtener objeto DataSource
String id = "db2adm";                 // Definir ID de usuario
String pw = "db2adm";                 // Definir contraseña
Connection con = ds.getConnection(id, pw);
                                      // Crear conexión
```

Como alternativa, si crea y despliega el objeto `DataSource`, puede establecer el ID de usuario y la contraseña invocando los métodos `DataSource.setUser` y `DataSource.setPassword` después de crear el objeto `DataSource`. Opcionalmente, puede invocar el método `DataSource.setSecurityMechanism` para indicar que está utilizando la seguridad basada en un ID de usuario y contraseña. Por ejemplo:

```
...
com.ibm.db2.jcc.DB2SimpleDataSource db2ds =    // Crear objeto DB2SimpleDataSource
  new com.ibm.db2.jcc.DB2SimpleDataSource();
db2ds.setDriverType(4);               // Definir tipo de controlador
db2ds.setDatabaseName("san_jose");   // Definir la ubicación
db2ds.setServerName("mvs1.sj.ibm.com");  // Definir nombre de servidor
db2ds.setPortNumber(5021);              // Definir el número de puerto
db2ds.setUser("db2adm");                // Definir el ID de usuario
db2ds.setPassword("db2adm");            // Definir la contraseña
db2ds.setSecurityMechanism(
  com.ibm.db2.jcc.DB2BaseDataSource.CLEAR_TEXT_PASSWORD_SECURITY);
                           // Definir el mecanismo de seguridad como
                           // ID de usuario y contraseña
```

Tareas relacionadas:

- "Conexión a una fuente de datos utilizando la interfaz DataSource" en la página 297
- "Creación y despliegue de objetos DataSource" en la página 338
- "Conexión a una fuente de datos utilizando la interfaz DriverManager con el Controlador JDBC de DB2 Universal" en la página 294

Información relacionada:

- "Propiedades del Controlador JDBC de DB2 Universal" en la página 400

Seguridad basada solo en el ID de usuario cuando se utiliza el Controlador JDBC de DB2 Universal

Para especificar la seguridad basada en un ID de usuario para una conexión JDBC, utilice una de las técnicas siguientes.

Para la interfaz DriverManager: establezca el ID de usuario y el mecanismo de seguridad definiendo las propiedades `user` y `securityMechanism` en un objeto `Properties`, y luego invoque la modalidad del método `getConnection` que hace uso del objeto `Properties` como parámetro. Por ejemplo:

```
import java.sql.*;        // Base JDBC
import com.ibm.db2.jcc.*;     // Implementación de JDBC 2.0 para DB2®
...
Properties properties = new Properties(); // Crear un objeto Properties
properties.put("user", "db2adm");  // Definir ID de usuario para conexión
properties.put("securityMechanism",
  new String("" + com.ibm.db2.jcc.DB2BaseDataSource.USER_ONLY_SECURITY + ""));
                 // Definir el mecanismo de seguridad como
                 // ID de usuario solamente
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                 // Definir URL para fuente de datos
Connection con = DriverManager.getConnection(url, properties);
                  // Crear la conexión
```

Para la interfaz `DataSource`: si crea y despliega el objeto `DataSource`, establezca el ID de usuario y el mecanismo de seguridad invocando los métodos `DataSource.setUser` y `DataSource.setSecurityMechanism` después de crear el objeto `DataSource`. Por ejemplo:

```
import java.sql.*;   // Base JDBC
import com.ibm.db2.jcc.*;     // Implementación de JDBC 2.0 para DB2
...
com.ibm.db2.jcc.DB2SimpleDataSource db2ds =
  new com.ibm.db2.jcc.DB2SimpleDataSource();
                                    // Crear objeto DB2SimpleDataSource
db2ds.setDriverType(4);             // Definir el tipo de controlador
db2ds.setDatabaseName("san_jose");  // Definir la ubicación
db2ds.setServerName("mvs1.sj.ibm.com");  // Definir el nombre de servidor
db2ds.setPortNumber(5021);              // Definir el número de puerto
db2ds.setUser("db2adm");                // Definir el ID de usuario
db2ds.setSecurityMechanism(
  com.ibm.db2.jcc.DB2BaseDataSource.USER_ONLY_SECURITY);
                           // Definir el mecanismo de seguridad como
                           // ID de usuario solamente
```

Seguridad mediante ID de usuario cifrado o contraseña cifrada con el Controlador JDBC de DB2 Universal

Si utiliza la seguridad mediante ID de usuario cifrado o contraseña cifrada al acceder a un servidor DB2® para z/OS™, es necesario tener habilitado Java™ Cryptography Extension, IBMJCE para z/OS en el servidor. Java Cryptography Extension forma parte de IBM® Developer Kit para OS/390®, Java 2 Technology Edition, o de IBM Developer Kit para z/OS, Java 2 Technology Edition. Para conocer cómo habilitar IBMJCE, vaya a este URL de la Web:

`http://www.ibm.com/servers/eserver/zseries/software/java/aboutj2.html`

Para especificar el mecanismo de seguridad basado en un ID de usuario cifrado o contraseña cifrada para una conexión JDBC, utilice una de las técnicas siguientes.

Para la interfaz `DriverManager`: establezca el ID de usuario, la contraseña y el mecanismo de seguridad definiendo las propiedades `user`, `password` y `securityMechanism` en un objeto `Properties`, y luego invoque la modalidad del método `getConnection` que hace uso del objeto `Properties` como parámetro. Por ejemplo, utilice un código como el siguiente para establecer el mecanismo de seguridad basado en un ID de usuario y una contraseña cifrada:

```
import java.sql.*;         // Base de JDBC
import com.ibm.db2.jcc.*;            // Implementación de JDBC 2.0 para DB2  ...
Properties properties = new Properties(); // Crear un objeto Properties
properties.put("user", "db2adm");         // Establecer ID usuario para la conexión
properties.put("password", "db2adm");     // Establecer contraseña para la conexión
properties.put("securityMechanism",
```

```
  new String("" + com.ibm.db2.jcc.DB2BaseDataSource.ENCRYPTED_PASSWORD_SECURITY +
  ""));
                                        // Establecer mecanismo de seguridad para
                                        // ID de usuario y contraseña cifrada
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                                       // Establecer URL de la fuente de datos
Connection con = DriverManager.getConnection(url, properties);
                                            // Crear la conexión
```

Para la interfaz DataSource: si crea y despliega el objeto DataSource, puede establecer el ID de usuario, la contraseña y el mecanismo de seguridad invocando los métodos DataSource.setUser, DataSource.setPassword y DataSource.setSecurityMechanism después de crear el objeto DataSource. Por ejemplo, utilice un código como el siguiente para establecer el mecanismo de seguridad basado en un ID de usuario cifrado y una contraseña cifrada:

```
import java.sql.*;    // Base JDBC
import com.ibm.db2.jcc.*; // Implementación de JDBC 2.0 para DB2
...
com.ibm.db2.jcc.DB2SimpleDataSource db2ds =
  new com.ibm.db2.jcc.DB2SimpleDataSource();
                                    // Crear el objeto DataSource
db2ds.setDriverType(4);             // Definir el tipo de controlador
db2ds.setDatabaseName("san_jose"); // Definir la ubicación
db2ds.setServerName("mvs1.sj.ibm.com");
                                  // Definir el nombre de servidor
db2ds.setPortNumber(5021);        // Definir el número de puerto
db2ds.setUser("db2adm");          // Definir el ID de usuario
db2ds.setPassword("db2adm");      // Definir la contraseña
db2ds.setSecurityMechanism(
  com.ibm.db2.jcc.DB2BaseDataSource.ENCRYPTED_PASSWORD_SECURITY);
                                  // Definir el mecanismo de seguridad
                                  // para el ID de usuario cifrado
                                  // y la contraseña cifrada
```

Tareas relacionadas:

- "Conexión a una fuente de datos utilizando la interfaz DataSource" en la página 297
- "Creación y despliegue de objetos DataSource" en la página 338
- "Conexión a una fuente de datos utilizando la interfaz DriverManager con el Controlador JDBC de DB2 Universal" en la página 294

Información relacionada:

- "Propiedades del Controlador JDBC de DB2 Universal" en la página 400

Seguridad Kerberos con el Controlador JDBC de DB2 Universal

El mecanismo de seguridad Kerberos solo está disponible para Conectividad de tipo 4 universal.

Si utiliza la seguridad Kerberos al acceder a un servidor DB2® para z/OS™, es necesario instalar y configurar los productos siguientes, o sus equivalentes:

- SecureWay® Security Server para z/OS y OS/390®
- OS/390 SecureWay Security Server Network Authentication and Privacy Service, que es un componente de OS/390 SecureWay Security Server.

  Esta es la implementación de IBM® OS/390 de Kerberos Versión 5.

  Para obtener más información, consulte el manual *OS/390 SecureWay Server Network Authentication and Privacy Service Administration*.

También es necesario habilitar los componentes siguientes del IBM Developer Kit para OS/390, Java™ 2 Technology Edition, o del IBM Developer Kit para z/OS, Java 2 Technology Edition:
- Java Cryptography Extension (IBMJCE) para OS/390
- IBM Java Generic Security Service (IBMJGSS)
- Java Authentication and Authorization Service (JAAS) para OS/390

Para conocer cómo habilitar esos componentes, vaya a este URL de la Web:
`http://www.ibm.com/servers/eserver/zseries/software/java/aboutj2.html`

Existen tres formas de especificar la seguridad Kerberos para una conexión:
- Con un ID de usuario y contraseña
- Sin un ID de usuario ni contraseña
- Con una credencial delegada

Utilización de la seguridad Kerberos con un ID de usuario y contraseña:

En este caso, Kerberos utiliza el ID de usuario y la contraseña especificados para obtener un certificado de concesión de certificados (TGT) que permite que el usuario se autentique ante el servidor DB2.

Es necesario que defina las propiedades `user`, `password`, `kerberosServerPrincipal` y `securityMechanism`. La propiedad `kerberosServerPrincipal` especifica la dirección del servidor Kerberos para la región en la que está registrado el cliente.

Para la interfaz `DriverManager`: establezca el ID de usuario, la contraseña, el servidor Kerberos y el mecanismo de seguridad definiendo las propiedades `user`, `password`, `kerberosServerPrincipal` y `securityMechanism` en un objeto `Properties`, y luego invoque la modalidad del método `getConnection` que hace uso del objeto `Properties` como parámetro. Por ejemplo, utilice un código como el siguiente para establecer el mecanismo de seguridad Kerberos con un ID de usuario y contraseña:

```
import java.sql.*;        // Base JDBC
import com.ibm.db2.jcc.*; // Implementación de JDBC 2.0 para DB2
...
Properties properties = new Properties(); // Crear un objeto Properties
properties.put("user", "db2adm");  // Definir ID de usuario para conexión
properties.put("password", "db2adm");  // Definir contraseña para conexión
properties.put("kerberosServerPrincipal", "kdcsrv1.sj.ibm.com");
                                  // Definir el servidor Kerberos
properties.put("securityMechanism",
  new String("" + com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY + ""));
                   // Definir el mecanismo de seguridad como Kerberos
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                   // Definir URL para fuente de datos
Connection con = DriverManager.getConnection(url, properties);
                  // Crear la conexión
```

Para la interfaz `DataSource`: si crea y despliega el objeto `DataSource`, establezca el servidor Kerberos y el mecanismo de seguridad invocando los métodos `DataSource.setKerberosServerPrincipal` y `DataSource.setSecurityMechanism` después de crear el objeto `DataSource`. Por ejemplo:

```
import java.sql.*;    // Base JDBC
import com.ibm.db2.jcc.*; // Implementación de JDBC 2.0 para DB2
...
com.ibm.db2.jcc.DB2SimpleDataSource db2ds =
  new com.ibm.db2.jcc.DB2SimpleDataSource();
                                  // Crear el objeto DataSource
db2ds.setDriverType(4);           // Definir el tipo de controlador
db2ds.setDatabaseName("san_jose"); // Definir la ubicación
```

```
db2ds.setUser("db2adm");           // Definir el ID de usuario
db2ds.setPassword("db2adm");       // Definir la contraseña
db2ds.setServerName("mvs1.sj.ibm.com");
                                   // Definir el nombre de servidor
db2ds.setPortNumber(5021);         // Definir el número de puerto
db2ds.setKerberosServerPrincipal("kdcsrv1.sj.ibm.com");
                                  // Definir el servidor Kerberos
db2ds.setSecurityMechanism(
  com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY);
                   // Definir el mecanismo de seguridad como Kerberos
```

Utilización de la seguridad Kerberos sin un ID de usuario ni contraseña:

En este caso, la antememoria de credenciales por omisión de Kerberos debe contener un certificado de concesión de certificados (TGT) para permitir que el usuario se autentique ante el servidor DB2.

Es necesario que defina las propiedades `kerberosServerPrincipal` y `securityMechanism`.

Para la interfaz `DriverManager`: establezca el servidor Kerberos y el mecanismo de seguridad definiendo las propiedades `kerberosServerPrincipal` y `securityMechanism` en un objeto `Properties`, y luego invoque la modalidad del método `getConnection` que hace uso del objeto `Properties` como parámetro. Por ejemplo, utilice un código como el siguiente para establecer el mecanismo de seguridad Kerberos sin un ID de usuario ni contraseña:

```
import java.sql.*;             // Base JDBC
import com.ibm.db2.jcc.*;     // Implementación de JDBC 2.0 para DB2
...
Properties properties = new Properties(); // Crear un objeto Properties
properties.put("kerberosServerPrincipal", "kdcsrv1.sj.ibm.com");
                                  // Definir el servidor Kerberos
properties.put("securityMechanism",
  new String("" + com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY + ""));
                   // Definir el mecanismo de seguridad como Kerberos
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                   // Definir URL para fuente de datos
Connection con = DriverManager.getConnection(url, properties);
                  // Crear la conexión
```

Para la interfaz `DataSource`: si crea y despliega el objeto `DataSource`, establezca el servidor Kerberos y el mecanismo de seguridad invocando los métodos `DataSource.setKerberosServerPrincipal` y `DataSource.setSecurityMechanism` después de crear el objeto `DataSource`. Por ejemplo:

```
import java.sql.*;  // Base JDBC
import com.ibm.db2.jcc.*;     // Implementación de JDBC 2.0 para DB2
...
DB2DataSource db2ds = new com.ibm.db2.jcc.DB2SimpleDataSource();
                                   // Crear el objeto DataSource
db2ds.setDriverType(4);            // Definir el tipo de controlador
db2ds.setDatabaseName("san_jose"); // Definir la ubicación
db2ds.setServerName("mvs1.sj.ibm.com");
                                   // Definir el nombre de servidor
db2ds.setPortNumber(5021);         // Definir el número de puerto
db2ds.setKerberosServerPrincipal("kdcsrv1.sj.ibm.com");
                                  // Definir el servidor Kerberos
db2ds.setSecurityMechanism(
  com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY);
                  // Definir el mecanismo de seguridad como Kerberos
```

Utilización de la seguridad Kerberos con una credencial delegada procedente de otro principal:

En este caso, el usuario se autentica ante el servidor DB2 utilizando una credencial delegada que le ha transferido otro principal.

Es necesario que defina las propiedades kerberosServerPrincipal, gssCredential y securityMechanism.

Para la interfaz DriverManager: defina el servidor Kerberos, la credencial delegada y el mecanismo de seguridad definiendo las propiedades kerberosServerPrincipal y securityMechanism en un objeto Properties. Debido a que la propiedad gssCredential no es una serie de caracteres, no puede utilizar el método Properties.put para definirla. En su lugar, utilice el método DB2BaseDataSource.setGSSCredential. Luego invoque la modalidad del método getConnection que hace uso del objeto Properties como parámetro. Por ejemplo, utilice un código como el siguiente para establecer el mecanismo de seguridad Kerberos sin un ID de usuario ni contraseña:

```
import java.sql.*;            // Base JDBC
import com.ibm.db2.jcc.*;     // Implementación de JDBC 2.0 para DB2
...
Properties properties = new Properties(); // Crear un objeto Properties
properties.put("kerberosServerPrincipal", "kdcsrv1.sj.ibm.com");
                                 // Definir el servidor Kerberos
properties.put("gssCredential",delegatedCredential);
                                // Definir la credencial delegada
properties.put("securityMechanism",
  new String("" + com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY + ""));

                  // Definir el mecanismo de seguridad como Kerberos
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                   // Definir URL para fuente de datos
Connection con = DriverManager.getConnection(url, properties);
                  // Crear la conexión
```

Para la interfaz DataSource: si crea y despliega el objeto DataSource, puede definir el servidor Kerberos, la credencial delegada y el mecanismo de seguridad invocando los métodos DataSource.setKerberosServerPrincipal, DataSource.setGssCredential y DataSource.setSecurityMechanism después de crear el objeto DataSource. Por ejemplo:

```
DB2DataSource db2ds = new com.ibm.db2.jcc.DB2SimpleDataSource();
                                  // Crear el objeto DataSource
db2ds.setDriverType(4);           // Definir el tipo de controlador
db2ds.setDatabaseName("san_jose"); // Definir la ubicación
db2ds.setServerName("mvs1.sj.ibm.com");  // Definir el nombre de servidor
db2ds.setPortNumber(5021);             // Definir el número de puerto
db2ds.setKerberosServerPrincipal("kdcsrv1.sj.ibm.com");
                                      // Definir el servidor Kerberos
db2ds.setGssCredential(delegatedCredential);
                                    // Definir la credencial delegada
db2ds.setSecurityMechanism(
  com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY);
                  // Definir el mecanismo de seguridad como Kerberos
```

Tareas relacionadas:

- "Conexión a una fuente de datos utilizando la interfaz DataSource" en la página 297
- "Creación y despliegue de objetos DataSource" en la página 338
- "Conexión a una fuente de datos utilizando la interfaz DriverManager con el Controlador JDBC de DB2 Universal" en la página 294

Información relacionada:

- "Propiedades del Controlador JDBC de DB2 Universal" en la página 400

Capítulo 20. Diagnóstico de problemas de JDBC y SQLJ

Las secciones siguientes contienen información sobre el diagnóstico problemas en JDBC y SQLJ.

Diagnóstico de problemas de JDBC y SQLJ cuando se utiliza el Controlador JDBC universal de DB2

Las secciones siguientes contienen información sobre el diagnóstico problemas en JDBC y SQLJ cuando se utiliza el Controlador JDBC universal de DB2.

Diagnóstico de problemas de JDBC y SQLJ con el Controlador JDBC de DB2 Universal

Para obtener datos a fin de diagnosticar problemas de SQLJ o JDBC con el Controlador JDBC universal de DB2, debe recoger datos de rastreo y ejecutar programas de utilidad para formatear los datos de rastreo. Debe ejecutar los programas de rastreo y diagnóstico solo bajo la dirección del soporte de software de IBM®.

Si su aplicación se conecta a un servidor DB2® UDB para z/OS™ o OS/390®, es necesario instalar en ese servidor varios procedimientos almacenados para poder recoger datos de rastreo. Esos procedimientos almacenados son también utilizados para algunas llamadas `DatabaseMetaData`. Los procedimientos almacenados son:

- SQLCOLPRIVILEGES
- SQLCOLUMNS
- SQLFOREIGNKEYS
- SQLGETTYPEINFO
- SQLPRIMARYKEYS
- SQLPROCEDURECOLS
- SQLPROCEDURES
- SQLSPECIALCOLUMNS
- SQLSTATISTICS
- SQLTABLEPRIVILEGES
- SQLTABLES
- SQLUDTS
- SQLCAMESSAGE

Para DB2 UDB para OS/390 y z/OS, Versión 7, o DB2 UDB para OS/390, Versión 6, los procedimientos almacenados se proporcionan en los PTF. Los PTF se pueden solicitar mediante los canales normales de servicio utilizando los números de PTF siguientes:

Tabla 79. PTF para DB2 Universal Database para z/OS y OS/390

Versión de DB2 Universal Database para z/OS y OS/390	**Número de PTF**
Versión 6	UQ72081 y UQ72082
Versión 7	UQ72083

Consulte al administrador del sistema DB2 UDB para z/OS para conocer si estos procedimientos almacenados están instalados.

Recogida de datos de rastreo de JDBC:

Utilice uno de los procedimientos siguientes para iniciar el rastreo:

Procedimiento 1:

1. Si utiliza la interfaz `DataSource` para conectar con una fuente de datos, invoque el método `DB2BaseDataSource.setTraceLevel` para definir el tipo de rastreo que necesite. El nivel de rastreo por omisión es `TRACE_ALL`. Consulte el tema Propiedades del Controlador JDBC de DB2 Universal para obtener información sobre cómo especificar más de un tipo de rastreo.
2. Invoque el método `DB2BaseDataSource.setJccLogWriter` para especificar el destino de rastreo y activar la función de rastreo.

Procedimiento 2:

Si utiliza la interfaz `DataSource` para conectar con una fuente de datos, invoque el método `javax.sql.DataSource.setLogWriter` para activar el rastreo. Con este método, `TRACE_ALL` es el único nivel de rastreo disponible.

Si utiliza la interfaz `DriverManager` para conectar con una fuente de datos, siga este procedimiento para iniciar el rastreo.

1. Invoque el método `DriverManager.getConnection` con la propiedad `traceLevel` establecida en el parámetro *info* o *url* correspondiente al tipo de rastreo que necesite. El nivel de rastreo por omisión es `TRACE_ALL`. Consulte el tema Propiedades del Controlador JDBC de DB2 Universal para conocer cómo especificar más de un tipo de rastreo.
2. Invoque el método `DriverManager.setLogWriter` para especificar el destino de rastreo y activar la función de rastreo.

Después de establecer una conexión, puede desactivar el rastreo o activarlo de nuevo, cambiar el destino de rastreo o cambiar el nivel de rastreo utilizando el método `DB2Connection.setJccLogWriter`. Para desactivar el rastreo, establezca el valor `logWriter` en `null`.

La propiedad `logWriter` es un objeto de tipo `java.io.PrintWriter`. Si su aplicación no puede manejar objetos `java.io.PrintWriter`, puede utilizar la propiedad `traceFile` para especificar el destino de los datos de salida de rastreo. Para utilizar la propiedad `traceFile`, establezca la propiedad `logWriter` en `null`, y establezca la propiedad `traceFile` en el nombre del archivo donde el controlador escribe los datos de rastreo. Este archivo y el directorio en el que reside deben tener acceso de escritura. Si el archivo ya existe, el controlador lo sobrescribe.

Procedimiento 3: si está utilizando la interfaz `DriverManager`, especifique las propiedades `traceFile` y `traceLevel` como parte del URL cuando cargue el controlador. Por ejemplo:

```
String url = "jdbc:db2://sysmvs1.stl.ibm.com:5021/san_jose" +
 ":traceFile=/u/db2p/jcctrace;" +
 "traceLevel=com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DRDA_FLOWS;";
```

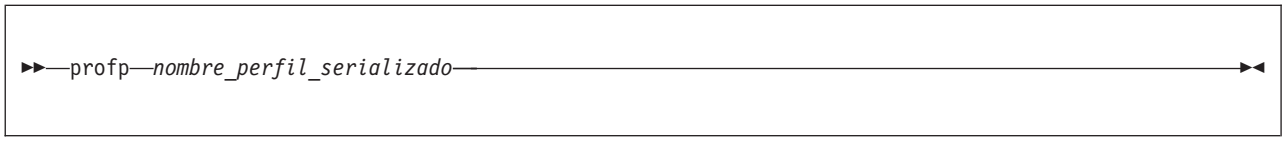
Programa de rastreo de ejemplo: para obtener un programa completo de ejemplo para el rastreo cuando se utiliza el Controlador JDBC universal de DB2, consulte el tema Ejemplo de rastreo cuando se utiliza el Controlador JDBC de DB2 Universal.

Recogida de datos de rastreo de SQLJ:

Para recoger datos de rastreo a fin de diagnosticar problemas durante el proceso de personalización o vinculación de SQLJ, especifique las opciones -tracelevel y -tracefile cuando ejecute el programa de utilidad db2sqljcustomize o db2sqljbind.

Formateo de información sobre un perfil serializado de SQLJ:

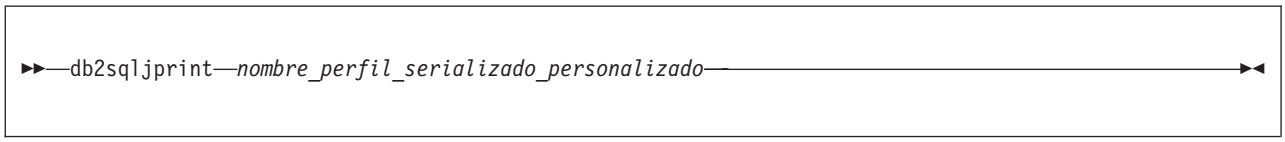
El programa de utilidad `profp` formatea información sobre cada cláusula SQLJ contenida en un perfil serializado. La sintaxis del programa de utilidad `profp` es:



Ejecute el programa de utilidad `profp` sobre el perfil serializado correspondiente a la conexión donde se produzca el error. Si se emite una excepción, se genera un rastreo de pila Java™. A partir del rastreo de pila, puede determinar qué perfil serializado se estaba utilizando cuando se emitió la excepción.

Formateo de información sobre un perfil serializado personalizado de SQLJ:

El programa de utilidad `db2sqljprint` formatea información sobre cada cláusula de SQLJ contenida en un perfil serializado que esté personalizado para el Controlador JDBC universal de DB2. La sintaxis del programa de utilidad `db2sqljprint` es:



Ejecute el programa de utilidad `db2sqljprint` sobre el perfil serializado personalizado correspondiente a la conexión donde se produzca el error.

Conceptos relacionados:

- "Ejemplo de rastreo para el Controlador JDBC de DB2 Universal" en la página 489

Información relacionada:

- "db2sqljcustomize - Mandato Personalizador de perfiles SQLJ de DB2" en la publicación *Consulta de mandatos*
- "Mandato db2sqljbind - DB2 SQLJ Profile Binder" en la publicación *Consulta de mandatos*
- "Propiedades del Controlador JDBC de DB2 Universal" en la página 400

Ejemplo de rastreo para el Controlador JDBC de DB2 Universal

El ejemplo siguiente muestra el uso de una clase para establecer una conexión y recoger y mostrar datos de rastreo cuando se utiliza el Controlador JDBC universal de DB2. La clase incluye un método para la interfaz `DriverManager` y un método para la interfaz `DataSource`.

```
public class TraceExample
{

  public static void main(String[] args)
  {
    sampleConnectUsingSimpleDataSource();
    sampleConnectWithURLUsingDriverManager();
  }

  private static void sampleConnectUsingSimpleDataSource()
  {
    java.sql.Connection c = null;
    java.io.PrintWriter printWriter =
     new java.io.PrintWriter(System.out, true);
                      // Visualiza en la consola.
                      // El valor true significa vaciado automático
                      // para no perder datos de rastreo
try {
      javax.sql.DataSource ds =
       new com.ibm.db2.jcc.DB2SimpleDataSource();
      ((com.ibm.db2.jcc.DB2BaseDataSource) ds).setServerName("sysmvs1.stl.ibm.com");
      ((com.ibm.db2.jcc.DB2BaseDataSource) ds).setPortNumber(5021);
      ((com.ibm.db2.jcc.DB2BaseDataSource) ds).setDatabaseName("san_jose");
      ((com.ibm.db2.jcc.DB2BaseDataSource) ds).setDriverType(4);

     ds.setLogWriter(printWriter);    // Esto activa el rastreo

     // Refinar el nivel de detalle del rastreo
     ((com.ibm.db2.jcc.DB2BaseDataSource) ds).
      setTraceLevel(com.ibm.db2.jcc.DB2SimpleDataSource.TRACE_CONNECTS |
      com.ibm.db2.jcc.DB2SimpleDataSource.TRACE_DRDA_FLOWS);

     // Esta petición de conexión se rastrea utilizando el
     // nivel de rastreo TRACE_CONNECTS | TRACE_DRDA_FLOWS
     c = ds.getConnection("myname", "mypass");

     // Cambiar el nivel de rastreo a TRACE_ALL
     // para todas las peticiones subsiguientes de la conexión
     ((com.ibm.db2.jcc.DB2Connection) c).setJccLogWriter(printWriter,
      com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL);
```

Figura 60. Ejemplo de rastreo para el Controlador JDBC universal de DB2 (Parte 1 de 5)

```
     // La sentencia INSERT siguiente se rastrea
          // utilizando el nivel de rastreo TRACE_ALL
     java.sql.Statement s1 = c.createStatement();
     s1.executeUpdate("INSERT INTO sampleTable(sampleColumn) VALUES(1)");
     s1.close();

      // El código siguiente inhabilita todo el rastreo de la conexión
     ((com.ibm.db2.jcc.DB2Connection) c).setJccLogWriter(null);

     // La sentencia INSERT siguiente no se rastrea
     java.sql.Statement s2 = c.createStatement();
     s2.executeUpdate("INSERT INTO sampleTable(sampleColumn) VALUES(1)");
     s2.close();

     c.close();
    }
    catch(java.sql.SQLException e) {
     com.ibm.db2.jcc.DB2ExceptionFormatter.printTrace(e,
      printWriter, "[TraceExample]");
    }
    finally {
      cleanup(c, printWriter);
      printWriter.flush();
    }
  }

  // Si el código se ha ejecutado satisfactoriamente, la
  // conexión debe estar ya cerrada. Compruebe si está cerrada.
  // Si la conexión está cerrada, simplemente finalice el programa.
  // Si se ha producido un error, intente retrotraer (rollback)
  // y cierre la conexión.

  private static void cleanup(java.sql.Connection c,
   java.io.PrintWriter printWriter)
  {
    if(c == null) return;

try {
     if(c.isClosed()) {
        printWriter.println("[TraceExample] " +
         "La conexión se ha cerrado satisfactoriamente");
         return;
     }

     // Si se llega a este punto, significa que se ha
     // producido un error.  Ejecute una retrotracción
     // y cierre la conexión.
     printWriter.println("[TraceExample] Se está retrotrayendo la conexión");
try {
       c.rollback();
     }
```

Figura 60. Ejemplo de rastreo para el Controlador JDBC universal de DB2 (Parte 2 de 5)

```
     catch(java.sql.SQLException e) {
       printWriter.println("[TraceExample] " +
        "Se capturó la siguiente java.sql.SQLException al intentar retrotraer:");
       com.ibm.db2.jcc.DB2ExceptionFormatter.printTrace(e, printWriter,
        "[TraceExample]");
       printWriter.println("[TraceExample] " +
        "No se puede retrotraer la conexión");
      }
      catch(java.lang.Throwable e) {
        printWriter.println("[TraceExample] Se capturó " +
         "la siguiente java.lang.Throwable al intentar retrotraer:");
        com.ibm.db2.jcc.DB2ExceptionFormatter.printTrace(e,
         printWriter, "[TraceExample]");
        printWriter.println("[TraceExample] No se puede " +
       "retrotraer la conexión");
      }

      // Cierre la conexión
     printWriter.println("[TraceExample] Se está cerrando la conexión");
try {
        c.close();
      }
      catch(java.sql.SQLException e) {
        printWriter.println("[TraceExample] Excepción al " +
         "intentar cerrar la conexión");
        printWriter.println("[TraceExample] Pueden producirse " +
         "puntos muertos si no se cierra la conexión.");
        com.ibm.db2.jcc.DB2ExceptionFormatter.printTrace(e, printWriter,
         "[TraceExample]");
      }
      catch(java.lang.Throwable e) {
        printWriter.println("[TraceExample] Se ha emitido Throwable " +
         "al intentar cerrar la conexión");
        printWriter.println("[TraceExample] Pueden producirse " +
         "puntos muertos si no se cierra la conexión.");
        com.ibm.db2.jcc.DB2ExceptionFormatter.printTrace(e, printWriter,
         "[TraceExample]");
      }
    }
    catch(java.lang.Throwable e) {
      printWriter.println("[TraceExample] No se puede " +
       "forzar el cierre de la conexión");
       printWriter.println("[TraceExample] Pueden producirse " +
       "puntos muertos si no se cierra la conexión.");
      com.ibm.db2.jcc.DB2ExceptionFormatter.printTrace(e, printWriter,
       "[TraceExample]");
    }
  }
```

Figura 60. Ejemplo de rastreo para el Controlador JDBC universal de DB2 (Parte 3 de 5)

```
  private static void sampleConnectWithURLUsingDriverManager()
  {
    java.sql.Connection c = null;

    // Esta vez, enviar printWriter a un archivo.
    java.io.PrintWriter printWriter = null;
try {
      printWriter =
        new java.io.PrintWriter(
          new java.io.BufferedOutputStream(
            new java.io.FileOutputStream("/temp/driverLog.txt"), 4096), true);
    }
    catch(java.io.FileNotFoundException e) {
      java.lang.System.err.println ("No se puede definir un transcriptor de impresión para el rastreo");
      java.lang.System.err.flush();
         return;
    }

try {
      Class.forName("com.ibm.db2.jcc.DB2Driver");
    }
    catch(ClassNotFoundException e) {
      printWriter.println("[TraceExample] Conectividad de tipo 4 universal " +
       "no está en classpath de la aplicación. No se puede cargar el controlador.");
      printWriter.flush();
         return;
    }

    // Este URL describe la fuente de datos de destino para la
    // conectividad de Tipo 4. La propiedad traceLevel se define
    // mediante la sintaxis del URL, y el rastreo del controlador
    // se envía al archivo "/temp/driverLog.txt"
    String databaseURL =
     "jdbc:db2://sysmvs1.stl.ibm.com:5021" +
     "/sample:traceFile=/temp/driverLog.txt;traceLevel=" +
     "(com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DRDA_FLOWS " +
     "| com.ibm.db2.jcc.DB2BaseDataSource.TRACE_CONNECTS);";

    // Definir otras propiedades
    java.util.Properties properties = new java.util.Properties();
    properties.setProperty("user", "myname");
    properties.setProperty("password", "mypass");
```

Figura 60. Ejemplo de rastreo para el Controlador JDBC universal de DB2 (Parte 4 de 5)

```
    try {
     // Esta petición de conexión se rastrea utilizando el
     // nivel de rastreo TRACE_CONNECTS | TRACE_DRDA_FLOWS
     c = java.sql.DriverManager.getConnection(databaseURL, properties);

     // Cambie a TRACE_ALL el nivel de rastreo para todas
     // las peticiones subsiguientes de la conexión
     ((com.ibm.db2.jcc.DB2Connection) c).setJccLogWriter(printWriter,
      com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL);

          // La sentencia INSERT siguiente se rastrea
          // utilizando el nivel de rastreo TRACE_ALL
     java.sql.Statement s1 = c.createStatement();
     s1.executeUpdate("INSERT INTO sampleTable(sampleColumn) VALUES(1)");
     s1.close();

     // Inhabilite todo el rastreo de la conexión
     ((com.ibm.db2.jcc.DB2Connection) c).setJccLogWriter(null);

      // El siguiente código de inserción de SQL no se rastrea
      java.sql.Statement s2 = c.createStatement();
      s2.executeUpdate("insert into sampleTable(sampleColumn) values(1)");
      s2.close();

      c.close();
    }
    catch(java.sql.SQLException e) {
     com.ibm.db2.jcc.DB2ExceptionFormatter.printTrace(e, printWriter,
      "[TraceExample]");
    }
    finally {
      cleanup(c, printWriter);
      printWriter.flush();
    }
  }
}
```

Figura 60. Ejemplo de rastreo para el Controlador JDBC universal de DB2 (Parte 5 de 5)

Tareas relacionadas:

- "Conexión a una fuente de datos utilizando la interfaz DataSource" en la página 297
- "Conexión a una fuente de datos utilizando la interfaz DriverManager con el Controlador JDBC de DB2 Universal" en la página 294

Información relacionada:

- "Propiedades del Controlador JDBC de DB2 Universal" en la página 400

Diagnóstico de problemas de JDBC y SQLJ cuando se utiliza el Controlador JDBC de DB2 de tipo 2

Las secciones siguientes contienen información sobre el diagnóstico de problemas de JDBC y SQLJ cuando se utiliza el Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows (Controlador JDBC de DB2 de tipo 2).

Recurso de rastreo de CLI/ODBC/JDBC

Este tema trata los siguientes aspectos:

- "Configuración del rastreo para CLI de DB2 y JDBC de DB2" en la página 495

- "Opciones de rastreo de CLI de DB2 y el archivo db2cli.ini" en la página 496
- "Opciones de rastreo de JDBC de DB2 y el archivo db2cli.ini" en la página 497
- "Rastreo del controlador CLI de DB2 y rastreo del gestor de controladores de ODBC" en la página 499
- "Controlador CLI de DB2, Controlador JDBC de tipo 2 basado en CLI y rastreos DB2" en la página 499
- "Rastreos de CLI de DB2 y de JDBC de DB2, y procedimientos almacenados de CLI o Java" en la página 500

La CLI de DB2 y el Controlador JDBC de tipo 2 basado en CLI para Linux, UNIX® y Windows® proporcionan recursos completos de rastreo. Por omisión, estos recursos están inhabilitados y no utilizan ningún recurso de proceso adicional. Cuando están habilitados, los recursos de rastreo generan uno o más archivos de registro de texto cada vez que una aplicación accede al controlador adecuado (CLI de DB2 o Controlador JDBC de tipo 2 basado en CLI). Estos archivos de registro proporcionan información detallada sobre:

- el orden en que la aplicación ha llamado a las funciones CLI o JDBC
- el contenido de los parámetros de entrada y salida que se han pasado a las funciones CLI o JDBC y se han recibido de las mismas
- los códigos de retorno y los mensajes de error o de aviso generados por las funciones CLI o JDBC

El análisis de los archivos de rastreo de CLI de DB2 y de JDBC de DB2® puede ayudar a los programadores de aplicaciones de varias formas. En primer lugar, los errores sutiles de lógica del programa y de inicialización de parámetros suelen ser evidentes en los rastreos. En segundo lugar, los rastreo de CLI de DB2 y JDBC de DB2 pueden sugerir formas de ajustar una aplicación o las bases de datos a las que accede. Por ejemplo, si un rastreo de CLI de DB2 muestra una tabla que se consulta varias veces en un determinado grupo de atributos, se puede crear un índice correspondiente a dichos atributos en la tabla para mejorar el rendimiento de la aplicación. Finalmente, el análisis de los archivos de rastreo de CLI de DB2 y JDBC de DB2 pueden ayudar a los programadores de aplicaciones a comprender el comportamiento de una interfaz o aplicación de otro proveedor.

Configuración del rastreo para CLI de DB2 y JDBC de DB2:

Los parámetros de configuración de los recursos de rastreo de CLI de DB2 y JDBC de DB2 se leen del archivo de configuración de CLI de DB2 `db2cli.ini`. Por omisión, este archivo se encuentra en la vía de acceso `\sqllib` para la plataforma Windows y en la vía de acceso `/sqllib/cfg` para las plataformas UNIX. Puede alterar temporalmente la vía de acceso por omisión estableciendo la variable de entorno DB2CLIINIPATH. En la plataforma Windows, es posible que se encuentre un archivo `db2cli.ini` adicional en el directorio del perfil (o inicial) del usuario si hay fuentes de datos definidas por el usuario que se han definido mediante el Gestor de controladores de ODBC. Este archivo `db2cli.ini` alterará temporalmente el archivo por omisión.

Para ver los parámetros de configuración de rastreo actuales de `db2cli.ini` desde el procesador de línea de mandatos, emita el siguiente mandato:

```
db2 GET CLI CFG FOR SECTION COMMON
```

Hay tres formas de modificar el archivo `db2cli.ini` para configurar los recursos de rastreo de CLI de DB2 y de JDBC de DB2:

- utilizar el Asistente de configuración de DB2, si está disponible

- editar de forma manual el archivo db2cli.ini mediante un editor de texto
- emitir el mandato UPDATE CLI CFG desde el procesador de línea de mandatos

Por ejemplo, el siguiente mandato, emitido desde el procesador de línea de mandatos, actualiza el archivo db2cli.ini y habilita el recurso de rastreo JDBC:

```
db2 UPDATE CLI CFG FOR SECTION COMMON USING jdbctrace 1
```

Notas:

1. Normalmente, las opciones de configuración de rastreo CLI de DB2 y JDBC de DB2 sólo se leen desde el archivo de configuración db2cli.ini en el momento en que se inicializa una aplicación. Sin embargo, se puede utilizar una opción de rastreo especial de db2cli.ini, TraceRefreshInterval, para indicar un intervalo en el que se deben volver a leer las opciones específicas de rastreo de CLI de DB2 desde el archivo db2cli.ini.
2. El recurso de rastreo de CLI de DB2 también se puede configurar de forma dinámica estableciendo los atributos de entorno SQL_ATTR_TRACE y SQL_ATTR_TRACEFILE. Estos valores alterarán temporalmente los valores contenidos en el archivo db2cli.ini.

Importante: Inhabilite los recursos de rastreo de CLI de DB2 y JDBC de DB2 cuando no los necesite. El rastreo innecesario puede reducir el rendimiento de la aplicación y puede generar archivos de registro de rastro no deseados. DB2 no suprime los archivos de rastreo generados y agrega la nueva información de rastreo a los archivos de rastreo existentes.

Opciones de rastreo de CLI de DB2 y el archivo db2cli.ini:

Cuando una aplicación que utiliza el controlador de CLI de DB2 se empieza a ejecutar, el controlador comprueba la existencia de opciones del recurso de rastreo en la sección [COMMON] del archivo db2cli.ini. Estas opciones de rastreo son palabras clave de rastreo específicas que se establecen en determinados valores en el archivo db2cli.ini bajo la sección [COMMON].

Nota: Puesto que las palabras clave de rastreo de CLI de DB2 aparecen en la sección [COMMON] del archivo db2cli.ini, sus valores se aplican a todas las conexiones de bases de datos a través del controlador de CLI de DB2.

Las palabras clave de rastreo de CLI de DB2 que se pueden definir son:

- Trace
- TraceComm
- TraceErrImmediate
- TraceFileName
- TraceFlush
- TraceFlushOnError
- TraceLocks
- TracePathName
- TracePIDList
- TracePIDTID
- TraceRefreshInterval
- TraceStmtOnly
- TraceTime
- TraceTimeStamp

Nota: Las palabras clave de rastreo de CLI de DB2 sólo se leen del archivo db2cli.ini una sola vez, cuando se inicializa la aplicación, a no ser que esté definida la palabra clave TraceRefreshInterval. Si esta palabra clave está definida, las palabras clave Trace y TracePIDList se vuelven a leer del archivo db2cli.ini de acuerdo con el intervalo especificado y se aplican, según proceda, a la aplicación que se esté ejecutando en ese momento.

A continuación se muestra un ejemplo de configuración de rastreo del archivo db2cli.ini en el que se utilizan estas palabras clave y valores de CLI de DB2:

```
[COMMON]
trace=1
TraceFileName=\temp\clitrace.txt
TraceFlush=1
```

Notas:

1. Las palabras clave de rastreo de CLI NO son sensibles a mayúsculas y minúsculas. Sin embargo, puede que los valores de palabras claves correspondientes a nombres de archivos y de vías de acceso sean sensibles a mayúsculas y minúsculas en algunos sistemas operativos (como en UNIX).
2. Si la palabra clave de rastreo de CLI de DB2 o su valor asociado en el archivo db2cli.ini no son válidos, el recurso de rastreo de CLI de DB2 los pasará por alto y utilizará el valor por omisión correspondiente a dicha palabra clave de rastreo.

Opciones de rastreo de JDBC de DB2 y el archivo db2cli.ini:

Cuando empieza a ejecutarse una aplicación que hace uso del Controlador JDBC de tipo 2 basado en CLI, el controlador también comprueba la existencia de opciones de recurso de rastreo en el archivo db2cli.ini. Al igual que sucede con las opciones de rastreo de CLI de DB2, las opciones de rastreo de JDBC de DB2 se especifican como pares palabra clave/valor situados en la sección [COMMON] del archivo db2cli.ini.

Nota: Debido a que las palabras clave de rastreo de JDBC de DB2 aparecen en la sección [COMMON] del archivo db2cli.ini, sus valores se aplican a todas las conexiones de base de datos realizadas a través del Controlador JDBC de tipo 2 basado en CLI.

Las palabras clave de rastreo de JDBC de DB2 que se pueden definir son:

- JDBCTrace
- JDBCTracePathName
- JDBCTraceFlush

JDBCTrace = 0 | 1

La palabra clave JDBCTrace controla si las otras palabras clave de rastreo de JDBC de DB2 tienen algún efecto en la ejecución del programa. Si se establece JDBCTrace en su valor por omisión, 0, se inhabilita el recurso de rastreo de JDBC de DB2. Si se establece JDBCTrace en 1, se habilita el rastreo.

Por sí misma, la palabra clave JDBCTrace tiene poco efecto y no genera datos de rastreo a menos que también se especifique la palabra clave JDBCTracePathName.

JDBCTracePathName = <vía_de_rastreo_completamente_calificada>

El valor de JDBCTracePathName es la vía de acceso completamente calificada del directorio donde se escribe toda la información de rastreo de

JDBC de DB2. El recurso de rastreo de JDBC de DB2 intenta generar un nuevo archivo de registro de rastreo cada vez que se ejecuta una aplicación JDBC que hace uso del Controlador JDBC de tipo 2 basado en CLI. Si la aplicación tiene varias hebras, se generará un archivo de registro de rastreo independiente para cada hebra. Se utiliza de forma automática una concatenación del ID del proceso de aplicación, el número de secuencia de la hebra y una serie que identifica la hebra para nombrar los archivos de registro de rastreo. No hay ningún nombre de vía de acceso por omisión en el que se graben los archivos de registro de salida de rastreo de JDBC de DB2.

JDBCTraceFlush = 0 | 1
La palabra clave JDBCTraceFlush especifica la frecuencia con que la información de rastreo se escribe en el archivo de registro de rastreo de JDBC de DB2. Por omisión, JDBCTraceFlush tiene el valor 0 y cada archivo de registro de rastreo de JDBC de DB2 se mantiene abierto hasta que la aplicación o hebra rastreada termina de forma normal. Si la aplicación termina de forma anómala, se puede perder algo de la información de rastreo que no se escribió en el archivo de registro de rastreo.

Para asegurar la integridad y plenitud de la información de rastreo que se escribe en el archivo de registro de rastreo de JDBC de DB2, la palabra clave JDBCTraceFlush se puede establecer en 1. Una vez que se han escrito las entradas de rastreo en el archivo de registro de rastreo, el controlador JDBC de DB2 cierra el archivo y lo reabre, agregando nuevas entradas al final del archivo. Esto garantiza que no se pierde información de rastreo.

Nota: *Cada operación de cerrar y volver a abrir el archivo de registro de JDBC de DB2 genera una actividad general de entrada/salida significativa y puede reducir considerablemente el rendimiento de la aplicación*.

A continuación se muestra un ejemplo de configuración de rastreo del archivo `db2cli.ini` en el que se utilizan las palabras clave y valores de JDBC de DB2:

```
[COMMON]
jdbctrace=1
JdbcTracePathName=\temp\jdbctrace\
JDBCTraceFlush=1
```

Notas:

1. Las palabras clave de rastreo de JDBC NO son sensibles a mayúsculas y minúsculas. Sin embargo, puede que los valores de palabras claves correspondientes a nombres de archivos y de vías de acceso sean sensibles a mayúsculas y minúsculas en algunos sistemas operativos (como en UNIX).
2. Si una palabra clave de rastreo de JDBC de DB2 o su valor asociado en el archivo `db2cli.ini` no son válidos, el recurso de rastreo de JDBC de DB2 los pasa por alto y utiliza el valor por omisión correspondiente a esa palabra clave.
3. Al habilitar el rastreo de JDBC de DB2 no se habilita el rastreo de CLI de DB2. El Controlador JDBC de tipo 2 basado en CLI depende del controlador CLI de DB2 para acceder a la base de datos. Por lo tanto, puede que los programadores de Java™ también deseen habilitar el rastreo de CLI de DB2 para obtener información adicional sobre cómo las aplicaciones interactúan con la base de datos a través de las distintas capas de software. Las opciones de rastreo de JDBC de DB2 y de CLI de DB2 son independientes entre sí y se pueden especificar juntas en cualquier orden bajo la sección [COMMON] del archivo `db2cli.ini`.

Rastreo del controlador CLI de DB2 y rastreo del gestor de controladores de ODBC:

Es importante comprender las diferencias entre un rastreo del gestor de controladores de ODBC y un rastreo del controlador de CLI de DB2. Un rastreo del gestor de controladores de ODBC muestra las llamadas de funciones de ODBC realizadas por una aplicación ODBC a un gestor de controladores de ODBC. Por el contrario, un rastreo del controlador de CLI de DB2 muestra las llamadas de funciones realizadas por el gestor de controladores de ODBC al controlador de CLI de DB2 *en nombre de la aplicación*.

Un gestor de controladores de ODBC puede reenviar algunas llamadas de funciones directamente desde la aplicación al controlador de CLI de DB2. Sin embargo, el gestor de controladores de ODBC puede también retrasar o evitar el reenvío de algunas llamadas de funciones al controlador. El gestor de controladores de ODBC también puede modificar los argumentos de funciones de la aplicación o correlacionar funciones de la aplicación con otras funciones antes de reenviar la llamada al controlador de CLI de DB2.

Las razones para la intervención de llamadas de funciones de aplicación por parte del gestor de controladores de ODBC incluyen:

- Para las aplicaciones escritas con funciones de ODBC 2.0 que ya no se recomiendan en ODBC 3.0, las funciones antiguas se correlacionarán con funciones nuevas.
- Los argumentos de funciones de ODBC 2.0 que ya no se recomiendan en ODBC 3.0 se correlacionarán con argumentos equivalentes de ODBC 3.0.
- La biblioteca de cursores de Microsoft® correlacionará llamadas como `SQLExtendedFetch()` con varias llamadas a `SQLFetch()` y a otras funciones soportadas para conseguir el mismo resultado final.
- La agrupación de conexiones del gestor de controladores de ODBC generalmente diferirá las peticiones `SQLDisconnect()` (o las evitará si la conexión se reutiliza).

Por esta y por otras razones, puede que los programadores de aplicaciones consideren el rastreo del gestor de controladores de ODBC un complemento útil al rastreo del controlador de CLI de DB2.

Para obtener más información sobre cómo capturar e interpretar rastreo del gestor de controladores de ODBC, consulte la documentación del gestor de controladores de ODBC. En las plataformas Windows, consulte el manual Microsoft ODBC 3.0 Software Development Kit and Programmer's Reference, también disponible en línea en: http://www.msdn.microsoft.com/.

Controlador CLI de DB2, Controlador JDBC de tipo 2 basado en CLI y rastreos DB2:

Internamente, el Controlador JDBC de tipo 2 basado en CLI hace uso del controlador CLI de DB2 para acceder a la base de datos. Por ejemplo, el Controlador JDBC de tipo 2 basado en CLI puede correlacionar internamente el método getConnection() de Java con la función `SQLConnect()` de CLI de DB2. Como resultado, el rastreo de CLI de DB2 puede ser un complemento útil al rastreo de JDBC de DB2 para los programadores de Java.

El controlador CLI de DB2 utiliza muchas funciones internas y específicas de DB2 para realizar su trabajo. Estas llamadas de función internas y específicas de DB2 se registran en el recurso de rastreo de DB2. Los programadores de aplicaciones no

encontrarán útiles los rastreos de DB2, puesto que sólo están diseñados para ayudar al Servicio de IBM® en la determinación y resolución de problemas.

Rastreos de CLI de DB2 y de JDBC de DB2, y procedimientos almacenados de CLI o Java:

En todas las plataformas de estación de trabajo, se pueden utilizar los recursos de rastreo de CLI de DB2 y de JDBC de DB2 para realizar el rastreo de procedimientos almacenados de CLI de DB2 y de JDBC de DB2.

La mayor parte de la información y las instrucciones sobre rastreo de CLI de DB2 y de JDBC de DB2 proporcionada en anteriores secciones es genérica y se aplica por igual a aplicaciones y a procedimientos almacenados. Sin embargo, a diferencia de las aplicaciones, que son clientes de un servidor de bases de datos (y generalmente se ejecutan en una máquina distinta de la del servidor de bases de datos), los procedimientos almacenados se ejecutan en el servidor de bases de datos. Por lo tanto, debe seguir los siguientes pasos adicionales cuando efectúe un rastreo de procedimientos almacenados de CLI de DB2 o de JDBC de DB2:

- Asegúrese de que las opciones de palabras clave de rastreo están especificadas en el archivo `db2cli.ini` del servidor DB2.
- Si la palabra clave TraceRefreshInterval no tiene un valor positivo distinto de cero, asegúrese de que todas las palabras clave estén configuradas correctamente antes del inicio de la base de datos (es decir, cuando se emite el mandato db2start). Si se cambian los valores de rastreo mientras se está ejecutando el servidor de bases de datos se pueden producir resultados inesperados. Por ejemplo, si se cambia el valor de TracePathName mientras se está ejecutando el servidor, la próxima vez que se ejecute un procedimiento almacenado, algunos archivos de rastreo se escribirán en la nueva vía de acceso, mientras que otros se escribirán en la vía de acceso original. Para asegurar la coherencia, reinicie el servidor cada vez que se modifique una palabra clave de rastreo que no sea Trace ni TracePIDList.

Conceptos relacionados:

- "db2cli.ini initialization file" en la publicación *CLI Guide and Reference, Volume 1*
- "Archivos de rastreo de CLI y JDBC" en la página 500

Información relacionada:

- "SQLSetEnvAttr function (CLI) - Set environment attribute" en la publicación *CLI Guide and Reference, Volume 2*
- "db2trc - Mandato Rastrear" en la publicación *Consulta de mandatos*
- "Mandato GET CLI CONFIGURATION" en la publicación *Consulta de mandatos*
- "Mandato UPDATE CLI CONFIGURATION" en la publicación *Consulta de mandatos*
- "Miscellaneous variables" en la publicación *Administration Guide: Performance*
- "CLI/ODBC configuration keywords listing by category" en la publicación *CLI Guide and Reference, Volume 1*

Archivos de rastreo de CLI y JDBC

Las aplicaciones que acceden a controladores de CLI de DB2® o JDBC de DB2 pueden utilizar los recursos de rastreo de CLI de DB2 o JDBC de DB2. Estos programas de utilidad registran todas las llamadas de funciones realizadas por los controladores de CLI de DB2 o JDBC de DB2 en un archivo de registro que resulta

útil en la determinación de problemas. Este tema describe cómo acceder a estos archivos de registro que generan los recursos de rastreo y cómo interpretarlos:

- "Ubicación de los archivos de rastreo de CLI y JDBC"
- "Interpretación del archivo de rastreo de CLI" en la página 502
- "Interpretación del archivo de rastreo de JDBC" en la página 506

Ubicación de los archivos de rastreo de CLI y JDBC:

Si se ha utilizado la palabra clave TraceFileName en el archivo `db2cli.ini` para especificar un nombre de archivo completamente calificado, el archivo de registro de rastreo de CLI de DB2 estará en la ubicación especificada. Si se ha especificado un nombre de archivo relativo para el archivo de registro de rastreo de CLI de DB2, la ubicación de dicho archivo dependerá de lo que el sistema operativo considere que es la vía de acceso actual de la aplicación.

Nota: Si el usuario que ejecuta la aplicación no tiene suficiente autoridad para grabar en el archivo de registro de rastreo de la vía de acceso especificada, no se generará ningún archivo ni se proporcionará ningún aviso ni error.

Si se ha utilizado la palabra clave TracePathName o JDBCTracePathName, o ambas, en el archivo `db2cli.ini` para especificar directorios completamente calificados, los archivos de registro de rastreo de CLI de DB2 y JDBC de DB2 estarán en la ubicación especificada. Si se ha especificado un nombre de directorio relativo para cualquiera de estos directorios de rastreo, o para ambos, el sistema operativo determinará su ubicación según lo que considere que es la vía de acceso actual de la aplicación.

Nota: Si el usuario que ejecuta la aplicación no tiene suficiente autoridad para grabar archivos de rastreo en la vía de acceso especificada, no se generará ningún archivo ni se proporcionará ningún aviso ni error. Si la vía de acceso de rastreo especificada no existe, no se creará.

Los recursos de rastreo de CLI de DB2 y JDBC de DB2 utilizan de forma automática el ID de proceso y número de secuencia de hebra de la aplicación para nombrar los archivos de registro de rastreo cuando se han definido las palabras clave TracePathName y JDBCTracePathName. Por ejemplo, un rastreo de CLI de DB2 de una aplicación con tres hebras puede generar los siguientes archivos de registro de rastreo de CLI de DB2: `100390.0`, `100390.1`, `100390.2`.

De forma similar, un rastreo de JDBC de DB2 de una aplicación Java™ con dos hebras puede generar los siguientes archivos de registro de rastreo de JDBC: `7960main.trc`, `7960Thread-1.trc`.

Nota: Si el directorio de rastreo contiene archivos de registro de rastreo antiguos y nuevos, se puede utilizar la información de indicación horaria y fecha de los archivos para localizar los archivos de rastreo más recientes.

Si parece que no se ha creado ningún archivo de salida de rastreo de CLI de DB2 o JDBC de DB2:

- Compruebe que las palabras clave de configuración de rastreo estén correctamente establecidas en el archivo `db2cli.ini`. Una forma rápida de hacerlo consiste en emitir el mandato `db2 GET CLI CFG FOR SECTION COMMON` desde el procesador de línea de mandatos.
- Asegúrese de que la aplicación se vuelve a iniciar después de actualizar el archivo `db2cli.ini`. En concreto, los recursos de rastreo de CLI de DB2 y JDBC

de DB2 se vuelven a inicializar durante el arranque de la aplicación. Una vez inicializado, el recurso de rastreo de JDBC de DB2 no se puede volver a configurar. El recurso de rastreo de CLI de DB2 se puede reconfigurar durante la ejecución, pero sólo si se ha especificado debidamente la palabra clave TraceRefreshInterval antes del inicio de la aplicación.

Nota: Sólo las palabras clave Trace y TracePIDList de CLI de DB2 se pueden reconfigurar durante la ejecución. *Los cambios realizados en otras palabras clave de CLI de DB2, incluida TraceRefreshInterval, no tienen ningún efecto si no se reinicia la aplicación*.

- Si la palabra clave TraceRefreshInterval se ha especificado antes del inicio de la aplicación y la palabra clave Trace se ha establecido inicialmente en 0, asegúrese de que ha transcurrido suficiente tiempo para que el recurso de rastreo de CLI de DB2 haya podido volver a leer el valor de la palabra clave Trace.
- Si se utiliza la palabra clave TracePathName o JDBCTracePathName, o ambas, para especificar directorios de rastreo, asegúrese de que dichos directorios existen antes de iniciar la aplicación.
- Asegúrese de que la aplicación tiene acceso de grabación en el archivo de registro de rastreo o en el directorio de rastreo especificados.
- Compruebe la variable de entorno DB2CLIINIPATH. Si está establecida, los recursos de rastreo de CLI de DB2 y JDBC de DB2 esperan que el archivo `db2cli.ini` esté en la ubicación especificada por esta variable.
- Si la aplicación utiliza ODBC como interfaz con el controlador de CLI de DB2, compruebe que se ha llamado satisfactoriamente a una de estas funciones: `SQLConnect()`, `SQLDriverConnect()` o `SQLBrowseConnect()`. No se grabará ninguna entrada en los archivos de registro de rastreo de CLI de DB2 hasta que se haya establecido satisfactoriamente una conexión con la base de datos.

Interpretación del archivo de rastreo de CLI:

Los rastreos de CLI de DB2 siempre empiezan con una cabecera que identifica el ID de proceso y el ID de hebra de la aplicación que ha generado el rastreo, la hora en que ha comenzado el rastreo e información específica del producto, como el nivel de build de DB2 y la versión del controlador de CLI de DB2. Por ejemplo:

```
1   [ Process: 1227, Thread: 1024 ]
2   [ Date, Time:           01-27-2002 13:46:07.535211 ]
3   [ Product:              QDB2/LINUX 7.1.0 ]
4   [ Level Identifier:     02010105 ]
5   [ CLI Driver Version:   07.01.0000 ]
6   [ Informational Tokens: "DB2 v7.1.0","n000510","" ]
```

Nota: En los ejemplos de rastreo que se muestran en esta sección se han añadido números de línea a la izquierda del rastreo. Estos números de línea se han añadido para clarificar la explicación y *no* aparecen en el rastreo de CLI de DB2 real.

Inmediatamente después de la cabecera del rastreo, suele haber varias entradas de rastreo relacionadas con el entorno y la asignación e inicialización del descriptor de contexto de la conexión. Por ejemplo:

```
7   SQLAllocEnv( phEnv=&bffff684 )
8       ---> Time elapsed - +9.200000E-004 seconds

9   SQLAllocEnv( phEnv=0:1 )
10      <--- SQL_SUCCESS   Time elapsed - +7.500000E-004 seconds

11  SQLAllocConnect( hEnv=0:1, phDbc=&bffff680 )
```

```
12      ---> Time elapsed - +2.334000E-003 seconds

13  SQLAllocConnect( phDbc=0:1 )
14      <--- SQL_SUCCESS   Time elapsed - +5.280000E-004 seconds

15  SQLSetConnectOption( hDbc=0:1, fOption=SQL_ATTR_AUTOCOMMIT, vParam=0 )
16      ---> Time elapsed - +2.301000E-003 seconds

17  SQLSetConnectOption( )
18      <--- SQL_SUCCESS   Time elapsed - +3.150000E-004 seconds

19  SQLConnect( hDbc=0:1, szDSN="SAMPLE", cbDSN=-3, szUID="", cbUID=-3,
                          szAuthStr="", cbAuthStr=-3 )
20      ---> Time elapsed - +7.000000E-005 seconds
21  ( DBMS NAME="DB2/LINUX", Version="07.01.0000", Fixpack="0x22010105" )

22  SQLConnect( )
23      <--- SQL_SUCCESS   Time elapsed - +5.209880E-001 seconds
24  ( DSN=""SAMPLE"" )

25  ( UID=" " )

26  ( PWD="*" )
```

En el ejemplo de rastreo anterior, observe que hay dos entradas para cada llamada de función de CLI de DB2 (por ejemplo, líneas 19-21 y 22-26 para la llamada de función `SQLConnect()`). Esto siempre es así en los rastreos de CLI de DB2. La primera entrada muestra los valores de los parámetros de entrada que se han pasado a la llamada de función, mientras que la segunda entrada muestra los valores de parámetros de salida de función y el código de retorno que se ha devuelto a la aplicación.

El ejemplo de rastreo anterior muestra que la función `SQLAllocEnv()` ha asignado satisfactoriamente un descriptor de contexto de entorno (phEnv=0:1) en la línea 9. Este descriptor de contexto se ha pasado a la función `SQLAllocConnect()`, la cual ha asignado satisfactoriamente un descriptor de contexto de conexión de base de datos (phDbc=0:1) en la línea 13. Luego, se ha utilizado la función `SQLSetConnectOption()` para establecer el atributo SQL_ATTR_AUTOCOMMIT de la conexión de phDbc=0:1 en SQL_AUTOCOMMIT_OFF (vParam=0) en la línea 15. Finalmente, se ha llamado a la función `SQLConnect()` para conectar con la base de datos de destino (SAMPLE) en la línea 19.

En la entrada de rastreo de entrada de la función `SQLConnect()`, en la línea 21, se incluye el nivel de build y de FixPak del servidor de la base de datos de destino. Otra información que también puede aparecen en esta entrada de rastreo incluye palabras claves de la serie de conexión de entrada y las páginas de códigos del cliente y del servidor. Por ejemplo, supongamos que también apareciera la siguiente información en la entrada de rastreo de `SQLConnect()`:

```
( Application Codepage=819, Database  Codepage=819,
  Char Send/Recv Codepage=819, Graphic Send/Recv Codepage=819,
  Application Char Codepage=819, Application Graphic Codepage=819 )
```

Esto significaría que la aplicación y el servidor de bases de datos estarían utilizando la misma página de códigos (819).

La entrada de rastreo de retorno de la función `SQLConnect()` también contiene información de conexión importante (líneas 24-26 en el rastreo de ejemplo anterior). Información adicional que puede aparecer en la entrada de retorno incluye los valores de la palabra clave PATCH1 o PATCH2 que se aplican a la

conexión. Por ejemplo, si se hubiera especificado PATCH2=27,28 en el archivo db2cli.ini bajo la sección COMMON, también aparecería la siguiente línea en la entrada de retorno de SQLConnect():

```
( PATCH2="27,28" )
```

Después de las entradas de rastreo relacionadas con el entorno y con la conexión se encuentran las entradas de rastreo relacionadas con sentencias. Por ejemplo:

```
27  SQLAllocStmt( hDbc=0:1, phStmt=&bffff684 )
28      ---> Time elapsed - +1.868000E-003 seconds

29  SQLAllocStmt( phStmt=1:1 )
30      <--- SQL_SUCCESS   Time elapsed - +6.890000E-004 seconds

31  SQLExecDirect( hStmt=1:1, pszSqlStr="CREATE TABLE GREETING (MSG
                                  VARCHAR(10))", cbSqlStr=-3 )
32      ---> Time elapsed - +2.863000E-003 seconds
33  ( StmtOut="CREATE TABLE GREETING (MSG VARCHAR(10))" )

34  SQLExecDirect( )
35      <--- SQL_SUCCESS   Time elapsed - +2.387800E-002 seconds
```

En el ejemplo de rastreo anterior, se ha utilizado el descriptor de contexto de la conexión de base de datos (phDbc=0:1) para asignar un descriptor de contexto de sentencia (phStmt=1:1) en la línea 29. Luego se ha ejecutado una sentencia de SQL no preparada en dicho descriptor de contexto de sentencia en la línea 31. Si se ha definido la palabra clave TraceComm=1 en el archivo db2cli.ini, las entradas de rastreo de la llamada de función SQLExecDirect() mostrarán información adicional sobre la comunicación cliente-servidor, de esta manera:

```
SQLExecDirect( hStmt=1:1, pszSqlStr="CREATE TABLE GREETING (MSG
                            VARCHAR(10))", cbSqlStr=-3 )
    ---> Time elapsed - +2.876000E-003 seconds
( StmtOut="CREATE TABLE GREETING (MSG VARCHAR(10))" )

    sqlccsend( ulBytes - 232 )
    sqlccsend( Handle - 1084869448 )
    sqlccsend( ) - rc - 0, time elapsed - +1.150000E-004
    sqlccrecv( )
    sqlccrecv( ulBytes - 163 ) - rc - 0, time elapsed - +2.243800E-002

SQLExecDirect( )
    <--- SQL_SUCCESS   Time elapsed - +2.384900E-002 seconds
```

Observe la información adicional de llamada de funciones sqlccsend() y sqlccrecv() de esta entrada de rastreo. La información de la llamada sqlccsend() revela la cantidad de datos que se han enviado del cliente al servidor, cuánto ha durado la transmisión y el éxito de dicha transmisión (0 = SQL_SUCCESS). Luego la información de la llamada sqlccrecv() revela el tiempo que el cliente ha esperado una respuesta del servidor y la cantidad de datos incluidos en la respuesta.

A menudo aparecerán varios descriptores de contexto de sentencias en el rastreo de CLI de DB2. Si presta atención al identificador del descriptor de contexto de la sentencia, puede seguir fácilmente el método de ejecución de un descriptor de contexto de sentencia de forma independiente de los demás descriptores de contexto de sentencia que aparecen en el rastreo.

Los métodos de ejecución de sentencias que aparecen en el rastreo de CLI de DB2 suelen ser más complicados que el del ejemplo anterior. Por ejemplo:

```
36  SQLAllocStmt( hDbc=0:1, phStmt=&bffff684 )
37      ---> Time elapsed - +1.532000E-003 seconds
```

```
38  SQLAllocStmt( phStmt=1:2 )
39      <--- SQL_SUCCESS   Time elapsed - +6.820000E-004 seconds

40  SQLPrepare( hStmt=1:2, pszSqlStr="INSERT INTO GREETING VALUES ( ? )",
                cbSqlStr=-3 )
41      ---> Time elapsed - +2.733000E-003 seconds
42  ( StmtOut="INSERT INTO GREETING VALUES ( ? )" )

43  SQLPrepare( )
44      <--- SQL_SUCCESS   Time elapsed - +9.150000E-004 seconds

45  SQLBindParameter( hStmt=1:2, iPar=1, fParamType=SQL_PARAM_INPUT,
                      fCType=SQL_C_CHAR, fSQLType=SQL_CHAR, cbColDef=14,
                      ibScale=0, rgbValue=&080eca70, cbValueMax=15,
                      pcbValue=&080eca4c )
46      ---> Time elapsed - +4.091000E-003 seconds

47  SQLBindParameter( )
48      <--- SQL_SUCCESS   Time elapsed - +6.780000E-004 seconds

49  SQLExecute( hStmt=1:2 )
50      ---> Time elapsed - +1.337000E-003 seconds
51  ( iPar=1, fCType=SQL_C_CHAR, rgbValue="Hello World!!!", pcbValue=14,
      piIndicatorPtr=14 )

52  SQLExecute( )
53      <--- SQL_ERROR   Time elapsed - +5.951000E-003 seconds
```

En el ejemplo de rastreo anterior, se ha utilizado el descriptor de contexto de conexión de base de datos (phDbc=0:1) para asignar un segundo descriptor de contexto de sentencia (phStmt=1:2) en la línea 38. Luego se ha preparado una sentencia de SQL con un marcador de parámetros en dicho descriptor de contexto de sentencia en la línea 40. Luego, se ha vinculado un parámetro de entrada (iPar=1) del tipo de SQL adecuado (SQL_CHAR) con el marcador de parámetros en la línea 45. Finalmente, se ha ejecutado la sentencia en la línea 49. Observe que tanto el contenido como la longitud del parámetro de entrada (rgbValue=″Hello World!!!″, pcbValue=14) se muestran en el rastreo en la línea 51.

La función `SQLExecute()` falla en la línea 52. Si la aplicación llama a una función de CLI de DB2 de diagnóstico, como `SQLError()`, para diagnosticar la causa del error, dicha causa aparecerá en el rastreo. Por ejemplo:

```
54  SQLError( hEnv=0:1, hDbc=0:1, hStmt=1:2, pszSqlState=&bffff680,
              pfNativeError=&bfffee78, pszErrorMsg=&bffff280,
              cbErrorMsgMax=1024, pcbErrorMsg=&bfffee76 )
55      ---> Time elapsed - +1.512000E-003 seconds

56  SQLError( pszSqlState="22001", pfNativeError=-302, pszErrorMsg="[IBM][CLI
       Driver][DB2/LINUX] SQL0302N  El valor de una variable del lenguaje
       principal de la sentencia EXECUTE u OPEN es demasiado alto para su
       uso correspondiente.
       SQLSTATE=22001", pcbErrorMsg=157 )
57      <--- SQL_SUCCESS   Time elapsed - +8.060000E-004 seconds
```

El mensaje de error que se devuelve en la línea 56 contiene el código de error nativo de DB2 que se ha generado (SQL0302N), el sqlstate correspondiente a dicho código (SQLSTATE=22001) y una breve descripción del error. En este ejemplo, el origen del error es evidente: en la línea 49, la aplicación intenta insertar una serie de 14 caracteres en una columna definida como VARCHAR(10) en la línea 31.

Si la aplicación no responde a un código de retorno de error o de aviso de una función de CLI de DB2 llamando a una función de diagnóstico como `SQLError()`,

el mensaje de aviso o de error se tiene que grabar igualmente en el rastreo de CLI de DB2. Sin embargo, es posible que la ubicación de dicho mensaje en el rastreo no esté cerca de donde se ha producido realmente el error. Además, el rastreo indicará que la aplicación no ha recuperado el mensaje de error o de aviso. Por ejemplo, si no se recupera, es posible que el mensaje de error del ejemplo anterior no aparezca hasta más adelante y parezca que no esté relacionado con la llamada de función de CLI de DB2, como en el siguiente caso:

```
SQLDisconnect( hDbc=0:1 )
    ---> Time elapsed - +1.501000E-003 seconds
    sqlccsend( ulBytes - 72 )
    sqlccsend( Handle - 1084869448 )
    sqlccsend( ) - rc - 0, time elapsed - +1.080000E-004
    sqlccrecv( )
    sqlccrecv( ulBytes - 27 ) - rc - 0, time elapsed - +1.717950E-001
( Unretrieved error message="SQL0302N  El valor de una variable del lenguaje
  principal en la sentencia EXECUTE u OPEN es demasiado alto para su uso
  correspondiente. SQLSTATE=22001" )

SQLDisconnect( )
    <--- SQL_SUCCESS   Time elapsed - +1.734130E-001 seconds
```

La parte final de un rastreo de CLI de DB2 debe mostrar que la aplicación libera la conexión de base de datos y los descriptores de contexto de entorno que había asignado anteriormente en el rastreo. Por ejemplo:

```
58  SQLTransact( hEnv=0:1, hDbc=0:1, fType=SQL_ROLLBACK )
59      ---> Time elapsed - +6.085000E-003 seconds
60  ( ROLLBACK=0 )

61  SQLTransact( )
        <--- SQL_SUCCESS   Time elapsed - +2.220750E-001 seconds

62  SQLDisconnect( hDbc=0:1 )
63      ---> Time elapsed - +1.511000E-003 seconds

64  SQLDisconnect( )
65      <--- SQL_SUCCESS   Time elapsed - +1.531340E-001 seconds

66  SQLFreeConnect( hDbc=0:1 )
67      ---> Time elapsed - +2.389000E-003 seconds

68  SQLFreeConnect( )
69      <--- SQL_SUCCESS   Time elapsed - +3.140000E-004 seconds

70  SQLFreeEnv( hEnv=0:1 )
71      ---> Time elapsed - +1.129000E-003 seconds

72  SQLFreeEnv( )
73      <--- SQL_SUCCESS   Time elapsed - +2.870000E-004 seconds
```

Interpretación del archivo de rastreo de JDBC:

Los rastreos de JDBC de DB2 siempre comienzan con una cabecera que contiene información importante del sistema, como valores de variables clave de entorno, el nivel de JDK o JRE, el nivel del controlador de JDBC de DB2 y el nivel del build de DB2. Por ejemplo:

```
1   ============================================================
2   |    Trace beginning on 2002-1-28 7:21:0.19
3   ============================================================

4   System Properties:
5   ------------------
6   user.language = en
```

```
7   java.home = c:\Program Files\SQLLIB\java\jdk\bin\..
8   java.vendor.url.bug =
9   awt.toolkit = sun.awt.windows.WToolkit
10  file.encoding.pkg = sun.io
11  java.version = 1.1.8
12  file.separator = \
13  line.separator =
14  user.region = US
15  file.encoding = Cp1252
16  java.compiler = ibmjitc
17  java.vendor = IBM® Corporation
18  user.timezone = EST
19  user.name = db2user
20  os.arch = x86
21  java.fullversion = JDK 1.1.8 IBM build n118p-19991124 (JIT ibmjitc
                       V3.5-IBMJDK1.1-19991124)
22  os.name = Windows® NT
23  java.vendor.url = http://www.ibm.com/
24  user.dir = c:\Program Files\SQLLIB\samples\java
25  java.class.path =
          .:C:\Program Files\SQLLIB\lib;C:\Program Files\SQLLIB\java;
          C:\Program Files\SQLLIB\java\jdk\bin\
26  java.class.version = 45.3
27  os.version = 5.0
28  path.separator = ;
29  user.home = C:\home\db2user
30  ----------------------------------------
```

Nota: En los ejemplos de rastreo que se muestran en esta sección se han añadido números de línea a la izquierda del rastreo. Estos números de línea se han añadido para clarificar la explicación y *no* aparecen en el rastreo de JDBC de DB2 real.

Inmediatamente después de la cabecera de rastreo, suele haber varias entradas de rastreo relacionadas con la inicialización del entorno JDBC y el establecimiento de conexión de base de datos. Por ejemplo:

```
31  jdbc.app.DB2Driver -> DB2Driver() (2002-1-28 7:21:0.29)
32  | Loaded db2jdbc from java.library.path
33  jdbc.app.DB2Driver <- DB2Driver() [Time Elapsed = 0.01]

34  DB2Driver - connect(jdbc:db2:sample)

35  jdbc.app.DB2ConnectionTrace -> connect( sample, info, db2driver, 0, false )
                  (2002-1-28 7:21:0.59)
36  | 10: connectionHandle = 1
37  jdbc.app.DB2ConnectionTrace <- connect() [Time Elapsed = 0.16]

38  jdbc.app.DB2ConnectionTrace -> DB2Connection (2002-1-28 7:21:0.219)
39  | source = sample
40  | Connection handle = 1
41  jdbc.app.DB2ConnectionTrace <- DB2Connection
```

En el ejemplo de rastreo anterior, se ha efectuado una petición de carga del controlador de JDBC de DB2 en la línea 31. Esta petición ha resultado satisfactoria, tal como se notifica en la línea 33.

El recurso de rastreo de JDBC de DB2 utiliza clases Java específicas para capturar la información de rastreo. En el ejemplo de rastreo anterior, una de estas clases de rastreo, DB2ConnectionTrace, ha generado dos entradas de rastreo situadas en las líneas 35-37 y 38-41.

La línea 35 muestra que se invoca el método connect() y los parámetros de entrada de dicha llamada de método. La línea 37 muestra que la llamada al método

connect() ha resultado satisfactoria, mientras que la línea 36 muestra el parámetro de salida de dicha llamada (Connection handle = 1).

Después de las entradas relacionadas con la conexión se suelen encontrar las entradas relacionadas con sentencias en el rastreo de JDBC. Por ejemplo:

```
42  jdbc.app.DB2ConnectionTrace -> createStatement() (2002-1-28 7:21:0.219)
43  | Connection handle = 1
44  | jdbc.app.DB2StatementTrace -> DB2Statement( con, 1003, 1007 )
          (2002-1-28 7:21:0.229)
45  | jdbc.app.DB2StatementTrace <- DB2Statement() [Time Elapsed = 0.0]
46  | jdbc.app.DB2StatementTrace -> DB2Statement (2002-1-28 7:21:0.229)
47  | | Statement handle = 1:1
48  | jdbc.app.DB2StatementTrace <- DB2Statement
49  jdbc.app.DB2ConnectionTrace <- createStatement - Time Elapsed = 0.01

50  jdbc.app.DB2StatementTrace -> executeQuery(SELECT * FROM EMPLOYEE WHERE
                                  empno = 000010) (2002-1-28 7:21:0.269)
51  | Statement handle = 1:1
52  | jdbc.app.DB2StatementTrace -> execute2( SELECT * FROM EMPLOYEE WHERE
                                    empno = 000010 ) (2002-1-28 7:21:0.269)
52  | | jdbc.DB2Exception -> DB2Exception() (2002-1-28 7:21:0.729)
53  | | | 10: SQLError = [IBM][CLI Driver][DB2/NT] SQL0401N  Los tipos de datos de
              los operandos de la operación "=" no son compatibles.
              SQLSTATE=42818
54  | | |     SQLState = 42818
55  | | |     SQLNativeCode = -401
56  | | |     LineNumber = 0
57  | | |     SQLerrmc = =
58  | | jdbc.DB2Exception <- DB2Exception() [Time Elapsed = 0.0]
59  | jdbc.app.DB2StatementTrace <- executeQuery - Time Elapsed = 0.0
```

En las líneas 42 y 43, la clase DB2ConnectionTrace ha notificado que se ha llamado al método de JDBC createStatement() con el descriptor de contexto de conexión 1. Dentro de dicho método, se ha llamado al método interno DB2Statement(), tal como indica otra clase del recurso de rastreo de JDBC de DB2, DB2StatementTrace. Observe que esta llamada al método interno aparece 'anidada' en la entrada de rastreo. Las líneas 47-49 muestran que los métodos han resultado satisfactorios y que se ha asignado el descriptor de contexto de sentencia 1:1.

En la línea 50, se ha realizado una llamada al método de consulta de SQL en la sentencia 1:1, pero la llamada ha generado una excepción en la línea 52. El mensaje de error se notifica en la línea 53 y contiene el código de error nativo de DB2 que se ha generado (SQL0401N), el sqlstate correspondiente a dicho código (SQLSTATE=42818) y una breve descripción del error. En este ejemplo, el error se debe a que la columna EMPLOYEE.EMPNO está definida como CHAR(6) y no como un valor entero, tal como se supone en la consulta.

Conceptos relacionados:

- "Recurso de rastreo de CLI/ODBC/JDBC" en la página 494

Información relacionada:

- "Miscellaneous variables" en la publicación *Administration Guide: Performance*
- "Trace CLI/ODBC configuration keyword" en la publicación *CLI Guide and Reference, Volume 1*
- "TraceComm CLI/ODBC configuration keyword" en la publicación *CLI Guide and Reference, Volume 1*
- "TraceFileName CLI/ODBC configuration keyword" en la publicación *CLI Guide and Reference, Volume 1*

- "TracePathName CLI/ODBC configuration keyword" en la publicación *CLI Guide and Reference, Volume 1*
- "TracePIDList CLI/ODBC configuration keyword" en la publicación *CLI Guide and Reference, Volume 1*
- "TraceRefreshInterval CLI/ODBC configuration keyword" en la publicación *CLI Guide and Reference, Volume 1*

Capítulo 21. Java 2 Platform Enterprise Edition

Las secciones siguientes describen Java 2 Platform Enterprise Edition (J2EE).

Visión general de Java 2 Platform Enterprise Edition (J2EE)

En el entorno empresarial global actual, las organizaciones tienen que ampliar su alcance, reducir sus costes y reducir sus tiempos de respuesta, proporcionando servicios a los que puedan acceder fácilmente sus clientes, empleados, proveedores y otros socios empresariales. Estos servicios deben tener las siguientes características:

- Altamente disponibles, para ajustarse a los requisitos del entorno empresarial global
- Seguros, para proteger la privacidad de los usuarios y la integridad de la empresa
- Fiables y escalables, de modo que las transacciones empresariales resulten precisas y se procesen con rapidez

En la mayoría de los casos, estos servicios se suministran con la ayuda de aplicaciones de varios enlaces en las que cada enlace tiene un objetivo específico. Java™ 2 Platform Enterprise Edition reduce el coste y la complejidad de desarrollar estos servicios de varios enlaces, lo que da lugar a servicios que se pueden desplegar rápidamente y se pueden mejorar fácilmente según los requisitos de la empresa.

Java 2 Enterprise Edition consigue estas ventajas definiendo una arquitectura estándar que se suministra como los siguientes elementos:

- Java 2 Enterprise Edition Application Model, un modelo de aplicación estándar para desarrollar servicios de clientes ligeros de varios enlaces
- Java 2 Enterprise Edition Platform, una plataforma estándar para alojar aplicaciones Java 2 Enterprise Edition
- Java 2 Enterprise Edition Compatibility Test Suite para verificar que un producto Java 2 Enterprise Edition Platform cumple con el estándar de Java 2 Enterprise Edition Platform
- Java 2 Enterprise Edition Reference Implementation para demostrar las funciones de Java 2 Enterprise Edition y para proporcionar una definición operativa de la plataforma Java 2 Enterprise Edition

Conceptos relacionados:

- "Java 2 Platform Enterprise Edition" en la página 511

Java 2 Platform Enterprise Edition

Java™ 2 Platform Enterprise Edition proporciona el entorno de tiempo de ejecución para alojar aplicaciones Java 2 Enterprise Edition. El entorno de tiempo de ejecución define cuatro tipos de componentes a los que un producto Java 2 Enterprise Edition debe dar soporte:

- Los clientes de aplicaciones son programas en lenguaje de programación Java que suelen ser programas GUI que se ejecutan en un sistema de escritorio. Los clientes de aplicaciones tienen acceso a todas las funciones del enlace medio de Java 2 Enterprise Edition.
- Los componentes applets y GUI que normalmente se ejecutan en un navegador web pero que se pueden ejecutar en otras aplicaciones o dispositivos que den soporte al modelo de programación de applets.
- Los servlets, JavaServer Pages (JSP), filtros y receptores de sucesos de la web que se suelen ejecutar en un navegador web y que pueden responder a peticiones HTTP procedentes de clientes web. Los servlets, JSP y filtros se pueden utilizar para generar páginas HTML que constituyen la interfaz de usuario de una aplicación. También se pueden utilizar para generar XML o datos en otro formato que consumen otros componentes de la aplicación. Los servlets, las páginas creadas con tecnología JSP, los filtros y los receptores de sucesos de la web reciben conjuntamente en esta especificación el nombre *componentes de la web*. Las aplicaciones web constan de componentes de la web y de otros datos como páginas HTML.
- Los componentes Enterprise JavaBeans™ (EJB) se ejecutan en un entorno gestionado que da soporte a transacciones. Los Enterprise Beans suelen contener la lógica empresarial correspondiente a una aplicación Java 2 Enterprise Edition.

Los componentes de aplicaciones listados anteriormente se pueden dividir en tres categorías, según el modo en que se pueden desplegar y gestionar:

- Componentes que se despliegan, gestionan y ejecutan en un servidor Java 2 Enterprise Edition.
- Componentes que se despliegan y gestionan en un servidor Java 2 Enterprise Edition pero que se cargan en una máquina cliente y se ejecutan en la misma.
- Componentes cuyo despliegue y gestión no están completamente definidos por esta especificación. Los clientes de aplicaciones pueden encontrarse en esta categoría.

El soporte de tiempo de ejecución correspondiente a estos componentes se proporciona mediante *contenedores*.

Conceptos relacionados:

- "Contenedores de Java 2 Platform Enterprise Edition" en la página 512
- "Enterprise Java Beans" en la página 520

Contenedores de Java 2 Platform Enterprise Edition

Un contenedor proporciona una vista federada de las API subyacentes de Java™ 2 Platform Enterprise Edition a los componentes de la aplicación. Un producto Java 2 Platform Enterprise Edition típico proporcionará un contenedor para cada tipo de componente de la aplicación: contenedor de clientes de la aplicación, contenedor de applets, contenedor de web y contenedor de Enterprise Beans. Las herramientas de contenedor también comprenden los formatos de archivo para empaquetar los componentes de la aplicación para su despliegue.

La especificación necesita que estos contenedores proporcionen un entorno de tiempo de ejecución compatible con Java, según lo definido en la especificación J2SE de Java 2 Platform Enterprise Edition, Standard Edition V1.3.1. Esta especificación define un conjunto de servicios estándar a los que debe dar soporte cada producto Java 2 Enterprise Edition. Estos servicios estándar son:

- Servicio HTTP
- Servicio HTTPS
- API de transacciones Java
- Método de invocación remota
- IDL Java
- API JDBC
- Servicio de mensajes de Java
- Naming and Directory Interface de Java
- JavaMail
- Infraestructura de activación de JavaBeans™
- API Java para el análisis XML
- Arquitectura de conectores
- Servicio de autenticación y autorización de Java

Conceptos relacionados:

- "Java Naming and Directory Interface (JNDI)" en la página 513
- "Enterprise Java Beans" en la página 520

Servidor Java 2 Platform Enterprise Edition

Como elemento subyacente de un contenedor Java™ 2 Platform Enterprise Edition se encuentra el servidor del que forma parte el contenedor. Un proveedor de productos Java 2 Enterprise Edition suele implantar las funciones de servidor Java 2 Platform Enterprise Edition mediante una infraestructura existente de proceso de transacciones en combinación con la tecnología J2SE. Las funciones del cliente Java 2 Platform Enterprise Edition suelen estar integradas en la tecnología J2SE.

Nota: IBM® WebSphere® Application Server es un servidor que cumple las especificaciones de Java 2 Platform Enterprise Edition.

Requisitos de bases de datos de Java 2 Enterprise Edition

La plataforma Java™ 2 Enterprise Edition necesita una base de datos a la que se pueda acceder a través de la API JDBC para el almacenamiento de los datos de la empresa. Se puede acceder a la base de datos desde componentes de la web, Enterprise Beans y componentes cliente de la aplicación. No hace falta que se pueda acceder a la base de datos desde los applets.

Conceptos relacionados:

- Capítulo 14, "Introducción al soporte de aplicaciones Java", en la página 283

Java Naming and Directory Interface (JNDI)

JNDI permite que las aplicaciones basadas en la plataforma Java™ accedan a varios servicios de nomenclatura y directorio. Esto forma parte del conjunto de interfaces de programación de aplicaciones (API) de Java Enterprise. JNDI permite a los programadores crear aplicaciones portables habilitadas para distintos servicios de nomenclatura y directorio que incluyen sistemas de archivos, servicios de directorio como Lightweight Directory Access Protocol (LDAP), Novell Directory Services y Network Information System (NIS) y sistemas de objetos distribuidos como Common Object Request Broker Architecture (CORBA), Invocación a métodos remotos (RMI) de Java y Enterprise JavaBeans™ (EJB).

La API JNDI tiene dos partes: una interfaz de nivel de aplicación que utilizan los componentes de la aplicación para acceder a los servicios de nomenclatura y directorio y una interfaz de proveedor de servicios para conectar con un proveedor de un servicio de nomenclatura y directorio.

Gestión de transacciones Java

Java™ 2 Enterprise Edition simplifica la programación de aplicaciones para la gestión de transacciones distribuidas. Java 2 Enterprise Edition incluye soporte de transacciones distribuidas a través de dos especificaciones, API de transacciones Java y Servicio de transacciones Java (JTS). JTA es una API de alto nivel, independiente de la implementación e independiente del protocolo, que permite a las aplicaciones y a los servidores de aplicaciones acceder a transacciones. Además, JTA está siempre habilitada.

El Controlador JDBC universal de DB2 y el Controlador JDBC de DB2® de Tipo 2 para Linux, UNIX® y Windows® cumplen las especificaciones de JTA y JTS.

JTA especifica interfaces Java estándar entre un gestor de transacciones y las partes que intervienen en un sistema de transacciones distribuidas: el gestor de recursos, el servidor de aplicaciones y las aplicaciones transaccionales.

JTS especifica la implantación de un Gestor de transacciones que da soporte a JTA e implanta la correlación Java de la especificación Object Transaction Service (OTS) 1.1 de OMG al nivel situado por debajo de la API. JTS propaga transacciones mediante IIOP.

JTA y JTS permiten a los servidores de aplicaciones Java 2 Enterprise Edition evitar al desarrollador de componentes la tarea de gestión de transacciones. Los programadores pueden definir las propiedades transaccionales de la tecnología EJB basándose en componentes durante el diseño o despliegue mediante sentencias declarativas en el descriptor de despliegue. El servidor de aplicaciones se hace cargo de la responsabilidad de la gestión de transacciones.

En DB2 y en el entorno WebSphere® Application Server, WebSphere Application Server asume la función de gestor de transacciones y DB2 actúa como gestor de recursos. WebSphere Application Server implementa JTS y parte de JTA, y los controladores JDBC también implementan parte de JTA para que WebSphere Application Server y DB2 puedan proporcionar transacciones distribuidas coordinadas.

No es necesario configurar DB2 de modo que esté habilitado para JTA en el entorno de WebSphere Application Server, pues los controladores JDBC detectan automáticamente ese entorno.

El Controlador DB2 JDBC de tipo 2 proporciona dos clases `DataSource`:
- `COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource`
- `COM.ibm.db2.jdbc.DB2XADataSource`

El Controlador JDBC universal de DB2 proporciona estas dos clases `DataSource`:
- `com.ibm.db2.jcc.DB2ConnectionPoolDataSource`
- `com.ibm.db2.jcc.DB2XADataSource`

WebSphere Application Server proporciona conexiones DB2 agrupadas con bases de datos. Si la aplicación va a participar en una transacción distribuida, se debe utilizar la clase `COM.ibm.db2.jdbc.DB2XADataSource` al definir fuentes de datos DB2 dentro de WebSphere Application Server.

Para ver información detallada sobre cómo configurar WebSphere Application Server con DB2, consulte WebSphere Application Server InfoCenter en:
`http://www-4.ibm.com/software/webservers/appserv/library.html`

Ejemplo de una transacción distribuida que utiliza métodos de JTA

La mejor forma de mostrar la utilización de transacciones distribuidas es compararlas con transacciones locales. En las transacciones locales, una aplicación JDBC confirma los cambios hechos en una base de datos e indica el final de una unidad de trabajo en una de las formas siguientes:

- Invocando los métodos `Connection.commit` o `Connection.rollback` después de ejecutar una o más sentencias de SQL
- Invocando el método `Connection.setAutoCommit(true)` al inicio de la aplicación para que los cambios se confirmen después de cada sentencia de SQL

La Figura 61 muestra el código mediante el que se ejecutan transacciones locales.

```
con1.setAutoCommit(false);  // Desactivar la confirmación automática
// Ejecutar sentencias de SQL
...
con1.commit();              // Confirmar la transacción
// Ejecutar más sentencias de SQL
...
con1.rollback();            // Retrotraer la transacción
con1.setAutoCommit(true);   // Permitir la confirmación después
                            // de cada sentencia de SQL
...
// Ejecutar más sentencias de SQL, que se confirmarán
// automáticamente después de cada sentencia de SQL.
```

Figura 61. Ejemplo de transacción local

En cambio, las aplicaciones que intervienen en transacciones distribuidas no pueden invocar los métodos `Connection.commit`, `Connection.rollback`, ni `Connection.setAutoCommit(true)` dentro de la transacción distribuida. En las transacciones distribuidas, los métodos `Connection.commit` o `Connection.rollback` no indican límites de transacción. En lugar de ello, las aplicaciones dejan que el servidor de aplicaciones gestione los límites de transacción. Normalmente las transacciones distribuidas suponen varias conexiones con una misma fuente de datos o con fuentes de datos diferentes, que pueden ser de fabricantes diversos.

La Figura 62 en la página 516 muestra una aplicación que utiliza transacciones distribuidas. Mientras se ejecuta el código mostrado en el ejemplo, el servidor de aplicaciones también está ejecutando otros EJB que forman parte de la misma transacción distribuida. Cuando todos los EJB han invocado `utx.commit()`, el servidor de aplicaciones confirma la transacción distribuida completa. Si cualquiera de los EJB no se ejecuta satisfactoriamente, el servidor de aplicaciones retrotrae todo el trabajo hecho por todos los EJB que están asociados a la transacción distribuida.

```
javax.transaction.UserTransaction utx;
// Utilice el método begin sobre un objeto UserTransaction
// para indicar el inicio de una transacción distribuida.
utx.begin();
...
// Ejecute sentencias de SQL con un objeto Connection.
// No invoque los métodos commit ni rollback de Connection.
...
// Utilice el método commit sobre el objeto UserTransaction
// para hacer que se confirmen todas las ramas de transacción
// e indicar el final de la transacción distribuida.

utx.commit();
...
```

Figura 62. Ejemplo de transacción distribuida cuando se utiliza un servidor de aplicaciones

La Figura 63 en la página 517 muestra un programa que utiliza métodos de JTA para ejecutar una transacción distribuida. Este programa actúa como gestor de transacciones y aplicación transaccional. Dos conexiones con dos fuentes de datos diferentes ejecutan tareas de SQL dentro de una sola transacción distribuida.

```
class XASample
{
  javax.sql.XADataSource xaDS1;
  javax.sql.XADataSource xaDS2;
  javax.sql.XAConnection xaconn1;
  javax.sql.XAConnection xaconn2;
  javax.transaction.xa.XAResource xares1;
  javax.transaction.xa.XAResource xares2;
  java.sql.Connection conn1;
  java.sql.Connection conn2;

  public static void main (String args []) throws java.sql.SQLException
  {
    XASample xat = new XASample();
    xat.runThis(args);
  }
  // En calidad de gestor de transacciones, este programa proporciona
  // el ID de transacción global y el calificador de rama. El ID de
  // transacción global y el calificador de rama no deben ser iguales
  // entre sí, y la combinación formada por ambos debe ser exclusiva
  // para este gestor de transacciones.
  public void runThis(String[] args)
  {
    byte[] gtrid = new byte[] { 0x44, 0x11, 0x55, 0x66 };
    byte[] bqual = new byte[] { 0x00, 0x22, 0x00 };
    int rc1 = 0;
    int rc2 = 0;

    try
    {

      javax.naming.InitialContext context = new javax.naming.InitialContext();
      /*
        * Observe la utilización de javax.sql.XADataSource en lugar
        * de utilizar una implementación de controlador específica,
        * tal como com.ibm.db2.jcc.DB2XADataSource, que solo se
        * puede utilizar para una conexión DB2.
        */
      xaDS1 = (javax.sql.XADataSource)context.lookup("checkingAccounts");
      xaDS2 = (javax.sql.XADataSource)context.lookup("savingsAccounts");

      // XADatasource contiene el ID de usuario y la contraseña.
      // Obtener el objeto XAConnection de cada XADataSource
      xaconn1 = xaDS1.getXAConnection();
      xaconn2 = xaDS2.getXAConnection();

      // Obtener el objeto java.sql.Connection de cada XAConnection
      conn1 = xaconn1.getConnection();
      conn2 = xaconn2.getConnection();

      // Obtener el objeto XAResource de cada XAConnection
      xares1 = xaconn1.getXAResource();
      xares2 = xaconn2.getXAResource();
```

Figura 63. Ejemplo de transacción distribuida que hace uso de la JTA (Parte 1 de 4)

```
      // Cree el objeto Xid de la transacción distribuida.
      // Este ejemplo utiliza la implementación com.ibm.db2.jcc.DB2Xid
      // de la interfaz Xid. Este Xid puede ser utilizado con cualquier
      // controlador JDBC que dé soporte a JTA.
      javax.transaction.xa.Xid xid1 =
                              new com.ibm.db2.jcc.DB2Xid(100, gtrid, bqual);

      // Inicie la transacción distribuida en las dos conexiones.
      // NO es necesario iniciar y finalizar las dos conexiones juntas.
      // Esto puede hacerse en hebras diferentes, junto con sus operaciones de SQL.
      xares1.start(xid1, javax.transaction.xa.XAResource.TMNOFLAGS);
      xares2.start(xid1, javax.transaction.xa.XAResource.TMNOFLAGS);
...
      // Ejecute las operaciones de SQL en la conexión 1.
      // Ejecute las operaciones de SQL en la conexión 2.
...
      // Ahora finalice la transacción distribuida en las dos conexiones.
      xares1.end(xid1, javax.transaction.xa.XAResource.TMSUCCESS);
      xares2.end(xid1, javax.transaction.xa.XAResource.TMSUCCESS);

      // Si el trabajo de la conexión 2 se ha realizado en otra hebra,
      // es necesaria aquí una llamada a thread.join() para esperar
      // a que termine el trabajo de la conexión 2.

    try
      { // Ahora prepare ambas ramas de la transacción distribuida.
        // Ambas ramas se deben preparar satisfactoriamente para
        // poder confirmar los cambios.
        // Si la transacción distribuida falla, se emite una
        // excepción XAException.
        rc1 = xares1.prepare(xid1);
        if(rc1 == javax.transaction.xa.XAResource.XA_OK)
        { // La preparación fue satisfactoria. Prepare la segunda conexión
          rc2 = xares2.prepare(xid1);
          if(rc2 == javax.transaction.xa.XAResource.XA_OK)
          { // Ambas conexiones se prepararon satisfactoriamente
            // y ninguna de ella era de solo lectura.
            xares1.commit(xid1, false);
            xares2.commit(xid1, false);
          }
          else if(rc2 == javax.transaction.xa.XAException.XA_RDONLY)
          { // La segunda conexión es de solo lectura, por lo que
            // solo se confirma la primera conexión.
            xares1.commit(xid1, false);
          }
        }
        else if(rc1 == javax.transaction.xa.XAException.XA_RDONLY)
        { // El SQL de la primera conexión es de solo lectura
          // (tal como un SELECT).
          // La preparación ha confirmado la conexión. Prepare la
          // segunda conexión.
          rc2 = xares2.prepare(xid1);
          if(rc2 == javax.transaction.xa.XAResource.XA_OK)
          { // La primera conexión es de solo lectura, pero la
            // segunda no lo es.
            // Confirme la segunda conexión.
            xares2.commit(xid1, false);
          }
          else if(rc2 == javax.transaction.xa.XAException.XA_RDONLY)
          { // Ambas conexiones son de solo lectura, y ambas
            // están ya confirmadas, por lo que no es necesaria
            // ninguna otra acción.
          }
        }
      }
```

Figura 63. Ejemplo de transacción distribuida que hace uso de la JTA (Parte 2 de 4)

```
  catch (javax.transaction.xa.XAException xae)
  { // La transacción distribuida ha fallado,
    // por lo que debe retrotraerla.
    // Notificar XAException para preparación/confirmación.
    System.out.println("Distributed transaction prepare/commit failed. " +
                       "Rolling it back.");
    System.out.println("XAException error code = " + xae.errorCode);
    System.out.println("XAException message = " + xae.getMessage());
    xae.printStackTrace();
try
    {
      xares1.rollback(xid1);
    }
    catch (javax.transaction.xa.XAException xae1)
    { // Notificar error de la retrotracción.
      System.out.println("distributed Transaction rollback xares1 failed");
      System.out.println("XAException error code = " + xae1.errorCode);
      System.out.println("XAException message = " + xae1.getMessage());
    }
try
    {
      xares2.rollback(xid1);
    }
    catch (javax.transaction.xa.XAException xae2)
    { // Notificar error de la retrotracción.
      System.out.println("distributed Transaction rollback xares2 failed");
      System.out.println("XAException error code = " + xae2.errorCode);
      System.out.println("XAException message = " + xae2.getMessage());
    }
  }

try
  {
    conn1.close();
    xaconn1.close();
  }
catch (Exception e)
  {
    System.out.println("Failed to close connection 1: " + e.toString());
    e.printStackTrace();
  }
try
  {
    conn2.close();
    xaconn2.close();
  }
catch (Exception e)
  {
    System.out.println("Failed to close connection 2: " + e.toString());
    e.printStackTrace();
  }
}
```

Figura 63. Ejemplo de transacción distribuida que hace uso de la JTA (Parte 3 de 4)

```
    catch (java.sql.SQLException sqe)
    {
      System.out.println("SQLException caught: " + sqe.getMessage());
      sqe.printStackTrace();
    }
    catch (javax.transaction.xa.XAException xae)
    {
      System.out.println("XA error is " + xae.getMessage());
      xae.printStackTrace();
    }
    catch (javax.naming.NamingException nme)
    {
      System.out.println(" Naming Exception: " + nme.getMessage());
    }
  }
}
```

Figura 63. Ejemplo de transacción distribuida que hace uso de la JTA (Parte 4 de 4)

Recomendación: Para lograr un mejor rendimiento, finalice una transacción distribuida antes de iniciar otra transacción distribuida o local.

Conceptos relacionados:

- "Gestión de transacciones Java" en la página 514

Enterprise Java Beans

La arquitectura Enterprise Java™ Beans es una arquitectura de componentes para el desarrollo y despliegue de aplicaciones de empresa distribuidas basadas en componentes. Las aplicaciones que se escriben utilizando la arquitectura Enterprise Java Beans se pueden escribir una sola vez y luego desplegar en cualquier plataforma servidor que dé soporte a la especificación Enterprise Java Beans. Las aplicaciones Java 2 Enterprise Edition implantan componentes de empresa del servidor mediante Enterprise Java Beans (EJB) que incluyen beans de sesión y beans de entidad.

Los beans de sesión representan servicios de empresa y no se comparten entre usuarios. Los beans de entidad son objetos transaccionales distribuidos, de múltiples usuarios, que representan datos permanentes. Los límites transaccionales de una aplicación EJB se pueden definir especificando transacciones gestionadas por contenedor o gestionadas por bean.

El programa de ejemplo `AccessEmployee.ear` utiliza Enterprise Java Beans para implementar una aplicación Java 2 Enterprise Edition para acceder a una base de datos DB2®. Encontrará este ejemplo en el directorio `SQLLIB/samples/websphere`.

La aplicación de ejemplo EJB proporciona dos servicios de empresa. Un servicio permite al usuario acceder a información sobre un empleado (que está almacenada en la tabla EMPLOYEE de la base de datos **sample**) mediante el número de empleado de dicho empleado. El otro servicio permite al usuario recuperar una lista de números de empleado de modo que el usuario pueda obtener un número de empleado para utilizarlo para consultar datos del empleado.

El siguiente ejemplo utiliza beans EJB para implementar una aplicación Java 2 Enterprise Edition para acceder a una base de datos DB2. En el ejemplo se utiliza la arquitectura Model-View-Controller (MVC), que es una arquitectura de GUI de uso habitual. Se utiliza la JSP para implementar la vista (el componente de

presentación). Un servlet actúa como controlador en el ejemplo. El servlet controla el flujo de trabajo y delega la petición del usuario al modelo, que se implementa mediante los EJB. El componente modelo del ejemplo consta de dos EJB: un bean de sesión y un bean de entidad. El bean de permanencia gestionada por contenedor (CMP), Employee, representa los objetos transaccionales distribuidos que representan los datos permanentes de la tabla EMPLOYEE de la base de datos sample. El término permanencia gestionada por contenedor significa que el contenedor EJB maneja todo el acceso a base de datos que necesita el bean de entidad. El código del bean no contiene ninguna llamada de acceso a base de datos (SQL). Como resultado, el código del bean no está enlazado a ningún mecanismo de almacenamiento permanente específico (base de datos). el bean de sesión, AccessEmployee, actúa como fachada del bean de entidad y proporciona una estrategia uniforme de acceso de clientes. Este diseño de fachada reduce el tráfico en la red entre el cliente EJB y el bean de entidad y resulta más eficiente en transacciones distribuidas que cuando el cliente EJB accede al bean de entidad directamente. El acceso a la base de datos DB2 se puede proporcionar desde el bean de sesión o el bean de entidad. Los dos servicios de la aplicación de ejemplo muestran ambos métodos para acceder a la base de datos DB2. En el primer servicio, se utiliza el bean de entidad:

```
//================================================
// Este método devuelve información sobre un empleado mediante la
// interacción con el bean de entidad identificado por el número
// de empleado proporcionado
public EmployeeInfo getEmployeeInfo(String empNo)
throws java.rmi.RemoteException
}
Employee employee = null;
try
}
employee = employeeHome.findByPrimaryKey(new EmployeeKey(empNo));
EmployeeInfo empInfo = new EmployeeInfo(empNo);
//establecer la información del empleado en el objeto de valor dependiente
empInfo.setEmpno(employee.getEmpno());
empInfo.setFirstName (employee.getFirstName());
empInfo.setMidInit(employee.getMidInit());
empInfo.setLastName(employee.getLastName());
empInfo.setWorkDept(employee.getWorkDept());
empInfo.setPhoneNo(employee.getPhoneNo());
empInfo.setHireDate(employee.getHireDate());
empInfo.setJob(employee.getJob());
empInfo.setEdLevel(employee.getEdLevel());
empInfo.setSex(employee.getSex());
empInfo.setBirthDate(employee.getBirthDate());
empInfo.setSalary(employee.getSalary());
empInfo.setBonus(employee.getBonus());
empInfo.setComm(employee.getComm());
return empInfo;
}
catch (java.rmi.RemoteException rex)
{
......
```

En el segundo servicio, que visualiza números de empleado, el bean de sesión, AccessEmployee, accede directamente a la base de datos sample de DB2.

```
/============================================
* Obtener la lista de números de empleado.
* @return Collection
*/
public Collection getEmpNoList()
{
ResultSet rs = null;
```

```
PreparedStatement ps = null;
Vector list = new Vector();
DataSource ds = null;
Connection con = null;
try
{
ds = getDataSource();
con = ds.getConnection();
String schema = getEnvProps(DBschema);
String query = "Select EMPNO from " + schema + ".EMPLOYEE";
ps = con.prepareStatement(query);
ps.executeQuery();
rs = ps.getResultSet();
EmployeeKey pk;
while (rs.next())
{
pk = new EmployeeKey();
pk.employeeId = rs.getString(1);
list.addElement(pk.employeeId);
}
rs.close();
return list;
```

Información relacionada:

- "Ejemplos de Java WebSphere" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Parte 5. Otras interfaces de programación

Capítulo 22. Programación en Perl

Consideraciones sobre la programación en Perl . . 525
Restricciones de Perl 525
Acceso a bases de datos de varias hebras en Perl 525
Conexiones de bases de datos en Perl 525
Captación de resultados en Perl 526
Marcadores de parámetros en Perl 527
Variables SQLSTATE y SQLCODE en Perl 527
Ejemplo de programa Perl 527

Consideraciones sobre la programación en Perl

Perl es un lenguaje de programación popular que está disponible libremente para muchos sistemas operativos. Mediante el controlador DBD::DB2, disponible en http://www.ibm.com/software/data/db2/perl, y el Módulo DBI (Perl Database Interface), disponible en http://www.perl.com, puede crear aplicaciones DB2® utilizando Perl.

Dado que Perl es un lenguaje interpretado y que el Módulo DBI de Perl utiliza SQL dinámico, Perl constituye el lenguaje ideal para crear y revisar con rapidez los prototipos de aplicaciones DB2. El Módulo DBI de Perl utiliza una interfaz que es muy parecida a las interfaces CLI y JDBC, lo cual facilita el transporte de los prototipos de Perl a CLI y JDBC.

La mayoría de proveedores de bases de datos proporcionan un controlador de base de datos para el Módulo DBI de Perl, lo cual significa que también se puede utilizar Perl para crear aplicaciones que accedan a datos de muchos servidores de bases de datos distintos. Por ejemplo, puede escribir en Perl una aplicación DB2 que conecte con una base de datos Oracle utilizando el controlador de base de datos DBD::Oracle, captar datos de la base de datos Oracle e insertar los datos en una base de datos DB2 utilizando el controlador de base de datos DBD::DB2.

Restricciones de Perl

El módulo DBI de Perl sólo soporta SQL dinámico. Cuando sea necesario ejecutar una sentencia varias veces, se puede mejorar el rendimiento de las aplicaciones DB2® en Perl emitiendo una llamada **prepare** para preparar la sentencia.

Para obtener información actual sobre las restricciones de la versión del controlador de DBD::DB2 que instale en la estación de trabajo, consulte el archivo CAVEATS contenido en el paquete del controlador de DBD::DB2.

Acceso a bases de datos de varias hebras en Perl

Perl no da soporte al acceso a bases de datos de varias hebras.

Conexiones de bases de datos en Perl

Para permitir que Perl cargue el módulo DBI, debe incluir la línea siguiente en la aplicación DB2®:

```
use DBI;
```

El módulo DBI carga automáticamente el controlador DBD::DB2 cuando se crea un *descriptor de contexto de base de datos* utilizando la sentencia **DBI->connect** con la sintaxis siguiente:

```
my $dbhandle = DBI->connect('dbi:DB2:dbalias', $userID, $password);
```

donde:

$dbhandle
representa el descriptor de contexto de base de datos devuelto por la sentencia de conexión

dbalias
representa un alias de DB2 catalogado en el directorio de bases de datos de DB2

$userID
representa el ID de usuario utilizado para conectar con la base de datos

$password
representa la contraseña para el ID de usuario utilizado para conectar con la base de datos

Captación de resultados en Perl

Puesto que el Módulo DBI de Perl sólo soporta SQL dinámico, no debe utilizar variables del lenguaje principal en las aplicaciones DB2 en Perl.

Procedimiento:

Para devolver resultados de una consulta de SQL, lleve a cabo los pasos siguientes:

1. Cree un descriptor de contexto de base de datos estableciendo conexión con la base de datos con la sentencia `DBI->connect`.
2. Cree un descriptor de contexto de sentencia a partir del descriptor de contexto de base de datos. Por ejemplo, puede llamar a **prepare** con una sentencia de SQL como argumento de serie para devolver el descriptor de contexto de sentencia *$sth*, tal como se muestra en la sentencia de Perl siguiente:

   ```
   my $sth = $descripbd->prepare(
      'SELECT firstnme, lastname
         FROM employee '
     );
   ```

3. Ejecute la sentencia de SQL llamando a **execute** en el descriptor de contexto de sentencia. Una llamada satisfactoria a **execute** asocia un conjunto de resultados al descriptor de contexto de sentencia. Por ejemplo, puede ejecutar la sentencia preparada en el ejemplo anterior utilizando la sentencia de Perl siguiente:

   ```
   #Nota: $rc representa el código de retorno de la llamada a execute
   my $rc = $sth->execute();
   ```

4. Capte una fila del conjunto de resultados asociado al descriptor de contexto de sentencia mediante una llamada a **fetchrow()**. El DBI de Perl devuelve una fila en forma de matriz con un valor por columna. Por ejemplo, puede devolver todas las filas del descriptor de contexto de sentencia del ejemplo anterior utilizando la sentencia de Perl siguiente:

   ```
   while (($firstnme, $lastname) = $sth->fetchrow()) {
      print "$firstnme $lastname\n";
   }
   ```

Conceptos relacionados:

• “Conexiones de bases de datos en Perl” en la página 525

Marcadores de parámetros en Perl

Para permitirle ejecutar una sentencia preparada utilizando distintos valores de entrada para campos específicos, el módulo DBI de Perl le permite preparar y ejecutar una sentencia utilizando marcadores de parámetros. Para incluir un marcador de parámetros en una sentencia de SQL, utilice un signo de interrogación (?).

El código Perl siguiente crea un descriptor de contexto de sentencia que acepta un marcador de parámetros para la cláusula WHERE de una sentencia SELECT. A continuación, el código ejecuta la sentencia dos veces, utilizando los valores de entrada 25000 y 35000 para sustituir el marcador de parámetros.

```
my $sth = $descripbd->prepare(
   'SELECT firstnme, lastname
      FROM employee
      WHERE salary > ?'
   );

my $rc = $sth->execute(25000);

⋮
my $rc = $sth->execute(35000);
```

Variables SQLSTATE y SQLCODE en Perl

Para devolver el SQLSTATE asociado a un descriptor de contexto de base de datos del DBI de Perl o a un descriptor de contexto de sentencia, llame al método **state**. Por ejemplo, para devolver el SQLSTATE asociado al descriptor de contexto de base de datos $descripbd, incluya la sentencia de Perl siguiente en la aplicación:

```
my $sqlstate = $descripbd->state;
```

Para devolver el SQLCODE asociado a un descriptor de contexto de base de datos del DBI de Perl o a un descriptor de contexto de sentencia, llame al método **err**. Para devolver el mensaje para un SQLCODE asociado a un descriptor de contexto de base de datos del DBI de Perl o a un descriptor de contexto de sentencia, llame al método **errstr**. Por ejemplo, para devolver el SQLCODE asociado al descriptor de contexto de base de datos $descripbd, incluya la sentencia de Perl siguiente en la aplicación:

```
my $sqlcode = $descripbd->err;
```

Ejemplo de programa Perl

A continuación se muestra un ejemplo de una aplicación escrita en Perl:

```
#!/usr/bin/perl
use DBI;

my $database='dbi:DB2:sample';
my $user='';
my $password='';

my $dbh = DBI->connect($database, $user, $password)
   or die "Can't connect to $database: $DBI::errstr";

my $sth = $dbh->prepare(
```

```
   q{ SELECT firstnme, lastname
      FROM employee }
   )
   or die "Can't prepare statement: $DBI::errstr";

my $rc = $sth->execute
   or die "Can't execute statement: $DBI::errstr";

print "Query will return $sth->{NUM_OF_FIELDS} fields.\n\n";
print "$sth->{NAME}->[0]: $sth->{NAME}->[1]\n";

while (($firstnme, $lastname) = $sth->fetchrow()) {
   print "$firstnme: $lastname\n";
}

# comprobar si hay problemas que puedan haber cancelado antes la captación
warn $DBI::errstr if $DBI::err;

$sth->finish;
$dbh->disconnect;
```

Capítulo 23. Programación en REXX

Consideraciones sobre la programación en REXX 529
Restricciones del lenguaje en REXX 530
Restricciones del lenguaje en REXX 530
Registro de SQLEXEC, SQLDBS y SQLDB2 en REXX 530
Acceso a bases de datos de varias hebras en REXX 531
Consideraciones sobre EUC en japonés o chino tradicional para REXX 531
SQL incorporado en aplicaciones REXX. 531
Variables del lenguaje principal en REXX 533
Variables del lenguaje principal en REXX . . . 533
Nombres de variables del lenguaje principal en REXX 534
Referencias a variables del lenguaje principal en REXX 534
Variables de indicador en REXX 534
Variables de REXX predefinidas 534
Variables del lenguaje principal de LOB en REXX 536
Sintaxis de las declaraciones de localizador de LOB en REXX 537
Sintaxis de las declaraciones de referencia de archivos LOB en REXX 537
Borrado de variables del lenguaje principal de LOB en REXX 538
Cursores en REXX. 539
Tipos de datos SQL soportados en REXX 539
Requisitos de ejecución para REXX 541
Creación y ejecución de aplicaciones REXX . . 541
Archivos de vinculación de REXX 542
Sintaxis de las API para REXX. 542
Llamada a procedimientos almacenados desde REXX 544
Procedimientos almacenados en REXX 544
Llamadas a procedimientos almacenados en REXX 544
Consideraciones del cliente para llamar a procedimientos almacenados en REXX 545
Consideraciones del servidor para llamar a procedimientos almacenados en REXX 546
Recuperación de valores de precisión y escala (SCALE) de campos decimales de la SQLDA . . 546

Consideraciones sobre la programación en REXX

En las secciones siguientes se explican las consideraciones especiales sobre la programación en el lenguaje principal. Se incluye información sobre la incorporación de sentencias de SQL, las restricciones del lenguaje y los tipos de datos soportados para las variables del lenguaje principal.

Nota: Soporte de REXX estabilizado en DB2 Versión 5, y no hay ninguna mejora de REXX planificada para el futuro. Por ejemplo, REXX no puede manejar identificadores de objetos SQL, como por ejemplo nombres de tabla, que tengan una longitud superior a 18 bytes. Para utilizar las características incorporadas en DB2 después de la Versión 5, como por ejemplo los nombres de tabla de longitud entre 19 y 128 bytes, debe escribir sus aplicaciones en un lenguaje que no sea REXX.

Puesto que REXX es un lenguaje interpretado, no se utiliza ningún precompilador, compilador ni enlazador. En su lugar, se utilizan tres API de DB2 para crear aplicaciones DB2 en REXX. Utilice dichas API para acceder a distintos elementos de DB2.

SQLEXEC
Da soporte al lenguaje SQL.

SQLDBS
Da soporte a las versiones de línea de mandatos de las API de DB2.

SQLDB2
Da soporte a una interfaz específica de REXX con el procesador de línea de mandatos. Consulte la descripción de la sintaxis de la API para REXX para ver detalles y restricciones sobre cómo se puede utilizar esta interfaz.

Conceptos relacionados:

- "Sintaxis de las API para REXX" en la página 542

Restricciones del lenguaje en REXX

Las siguientes secciones describen las restricciones del lenguaje correspondientes a REXX.

Restricciones del lenguaje en REXX

Es posible que los símbolos que se encuentren dentro de sentencias o mandatos que se pasen a las rutinas SQLEXEC, SQLDBS y SQLDB2 puedan corresponder a variables de REXX. En este caso, el intérprete de REXX sustituye el valor de la variable antes de llamar a SQLEXEC, SQLDBS o SQLDB2.

Para evitar esta situación, encierre las series de sentencias entre comillas (' ' o " "). Si no utiliza comillas, el intérprete de REXX resolverá los nombres de variable conflictivos, en lugar de que se pasen a las rutinas SQLEXEC, SQLDBS o SQLDB2.

En REXX/SQL no se soporta el SQL compuesto.

Los procedimientos almacenados de REXX/SQL se soportan en los sistemas operativos Windows®, pero no en AIX®.

Tareas relacionadas:

- "Registro de SQLEXEC, SQLDBS y SQLDB2 en REXX" en la página 530

Registro de SQLEXEC, SQLDBS y SQLDB2 en REXX

Antes de utilizar cualquiera de las API de DB2 o emitir sentencias de SQL en una aplicación, debe registrar las rutinas SQLDBS, SQLDB2 y SQLEXEC. Así se notifica al intérprete de REXX sobre los puntos de entrada de REXX/SQL. El método que utilice para realizar el registro variará ligeramente entre las plataformas basadas en Windows y AIX.

Procedimiento:

Utilice los ejemplos siguientes para ver la sintaxis correcta para registrar cada rutina:

Registro de ejemplo en plataformas basadas en Windows

```
/* ------------ Registrar SQLDBS con REXX  -------------------------*/
If Rxfuncquery('SQLDBS')  <> 0 then
    rcy = Rxfuncadd('SQLDBS','DB2AR','SQLDBS')
If rcy \= 0 then
    do
      say 'SQLDBS was not successfully added to the REXX environment'
     signal rxx_exit
   end

/* ------------ Registrar SQLDB2 con REXX  -------------------------*/
If Rxfuncquery('SQLDB2')  <> 0 then
    rcy = Rxfuncadd('SQLDB2','DB2AR','SQLDB2')
If rcy \= 0 then
    do
      say 'SQLDB2 was not successfully added to the REXX environment'
     signal rxx_exit
```

```
   end

/* ----------------- Registrar SQLEXEC con REXX  --------------------*/
If Rxfuncquery('SQLEXEC') <> 0 then
    rcy = Rxfuncadd('SQLEXEC','DB2AR','SQLEXEC')
If rcy \= 0 then
    do
      say 'SQLEXEC was not successfully added to the REXX environment'
     signal rxx_exit
   end
```

Ejemplo de registro en AIX

```
   /* ------------ Registrar SQLDBS, SQLDB2 y SQLEXEC con REXX --------*/
 rcy = SysAddFuncPkg("db2rexx")
 If rcy \= 0 then
   do
     say 'db2rexx was not successfully added to the REXX environment'
     signal rxx_exit
   end
```

En plataformas basadas en Windows, sólo es necesario ejecutar una vez el mandato RxFuncAdd para todas las sesiones.

En AIX, se debe ejecutar SysAddFuncPkg en cada aplicación REXX/SQL.

En la documentación de REXX para plataformas basadas en Windows y AIX, respectivamente, encontrará detalles sobre las API RXfuncadd y SysAddFuncPkg.

Acceso a bases de datos de varias hebras en REXX

REXX no da soporte al acceso a bases de datos de varias hebras.

Consideraciones sobre EUC en japonés o chino tradicional para REXX

Las aplicaciones REXX no se soportan en los entornos EUC en japonés o chino tradicional.

SQL incorporado en aplicaciones REXX

Las aplicaciones REXX utilizan API que les permiten utilizar la mayoría de las funciones que suministran las API del gestor de bases de datos y SQL. A diferencia de las aplicaciones escritas en un lenguaje compilado, las aplicaciones REXX no se precompilan. En su lugar, un manejador de SQL dinámico procesa todas las sentencias de SQL. Al combinar REXX con estas API que se pueden llamar, tiene acceso a la mayoría de las funciones del gestor de bases de datos. Aunque REXX no da soporte directamente a algunas API utilizando SQL incorporado, se puede acceder a las mismas utilizando el procesador de línea de mandatos de DB2® desde dentro de la aplicación REXX.

Como REXX es un lenguaje de interpretación, es posible que le resulte más fácil desarrollar y depurar los prototipos de aplicación en REXX que en lenguajes principales compilados. Aunque las aplicaciones DB2 codificadas en REXX no proporcionan el rendimiento de las aplicaciones DB2 que utilizan lenguajes compilados, proporcional la posibilidad de crear aplicaciones DB2 sin tener que precompilar, compilar, enlazar o utilizar software adicional.

Utilice la rutina SQLEXEC para procesar todas las sentencias de SQL. Los argumentos de serie de caracteres para la rutina SQLEXEC están formados por los elementos siguientes:
- Palabras clave de SQL
- Identificadores declarados previamente
- Variables del lenguaje principal de sentencia

Realice cada petición pasando una sentencia de SQL válida a la rutina SQLEXEC. Utilice la sintaxis siguiente:

```
CALL SQLEXEC 'sentencia'
```

Las sentencias de SQL se pueden continuar en más de una línea. Cada parte de la sentencia se debe encerrar entre comillas, y se debe delimitar mediante una coma el texto adicional de la sentencia, del modo siguiente:

```
CALL SQLEXEC 'SQL text',
             'additional text',
                  .
                  .
                  .
             'final text'
```

A continuación se muestra un ejemplo de incorporación de una sentencia de SQL en REXX:

```
statement = "UPDATE STAFF SET JOB = 'Clerk' WHERE JOB = 'Mgr'"
CALL SQLEXEC 'EXECUTE IMMEDIATE :statement'
IF ( SQLCA.SQLCODE < 0) THEN
   SAY 'Update Error:  SQLCODE = ' SQLCA.SQLCODE
```

En este ejemplo, el campo `SQLCODE` de la estructura `SQLCA` se comprueba para determinar si la actualización ha resultado satisfactoria.

Se aplican las normas siguientes a las sentencias de SQL incorporado:
- Las sentencias de SQL siguientes se pueden pasar directamente a la rutina SQLEXEC:
  - CALL
  - CLOSE
  - COMMIT
  - CONNECT
  - CONNECT TO
  - CONNECT RESET
  - DECLARE
  - DESCRIBE
  - DISCONNECT
  - EXECUTE
  - EXECUTE IMMEDIATE
  - FETCH
  - FREE LOCATOR
  - OPEN
  - PREPARE
  - RELEASE
  - ROLLBACK
  - SET CONNECTION

  Existen otras sentencias de SQL que se deben procesar dinámicamente utilizando las sentencias EXECUTE IMMEDIATE o PREPARE y EXECUTE junto con la rutina SQLEXEC.

- No se pueden utilizar variables del lenguaje principal en las sentencias CONNECT y SET CONNECTION en REXX.
- Los nombres de cursor y los nombres de sentencia están predefinidos del modo siguiente:

  de c1 a c100
  Nombres de cursor, que van de *c1* a *c50* para los cursores declarados sin la opción WITH HOLD, y de *c51* a *c100* para los cursores declarados utilizando la opción WITH HOLD.

  El identificador de nombre de cursor se utiliza para las sentencias DECLARE, OPEN, FETCH y CLOSE. Identifica el cursor utilizado en la petición de SQL.

  de s1 a s100
  Nombres de sentencia, que van de *s1* a *s100*.

  El identificador de nombre de sentencia se utiliza con las sentencias DECLARE, DESCRIBE, PREPARE y EXECUTE.

  Se deben utilizar identificadores declarados previamente para los nombres de cursor y de sentencia. No se admiten otros nombres.
- Cuando se declaren cursores, el nombre de cursor y el nombre de sentencia deben corresponder en la sentencia DECLARE. Por ejemplo, si se utiliza *c1* como nombre de cursor, se debe utilizar *s1* como nombre de sentencia.
- No utilice comentarios dentro de una sentencia de SQL.

Variables del lenguaje principal en REXX

Las secciones siguientes describen cómo declarar y utilizar variables del lenguaje principal en programas REXX.

Variables del lenguaje principal en REXX

Las variables del lenguaje principal son variables del lenguaje REXX a las que se hace referencia en las sentencias de SQL. Permiten que una aplicación pase datos de entrada a DB2 y reciba datos de salida de éste. No es necesario que las aplicaciones REXX declaren las variables del lenguaje principal, a excepción de las variables de localizadores de LOB y de referencia de archivos LOB. Los tamaños y tipos de datos de las variables de sistema principal se determinan en el tiempo de ejecución cuando se hace referencia a las variables. Las siguientes secciones describen las normas a seguir para dar nombre y utilizar variables del lenguaje principal.

Conceptos relacionados:
- “Nombres de variables del lenguaje principal en REXX” en la página 534
- “Referencias a variables del lenguaje principal en REXX” en la página 534
- “Variables de indicador en REXX” en la página 534
- “Variables del lenguaje principal de LOB en REXX” en la página 536
- “Borrado de variables del lenguaje principal de LOB en REXX” en la página 538

Información relacionada:
- “Variables de REXX predefinidas” en la página 534
- “Sintaxis de las declaraciones de localizador de LOB en REXX” en la página 537
- “Sintaxis de las declaraciones de referencia de archivos LOB en REXX” en la página 537

- "Tipos de datos SQL soportados en REXX" en la página 539

Nombres de variables del lenguaje principal en REXX

Cualquier variable de REXX correctamente denominada se puede utilizar como variable del lenguaje principal. Un nombre de variable puede tener una longitud máxima de 64 caracteres. No termine el nombre con un punto. Un nombre de variable del lenguaje principal puede constar de caracteres alfabéticos, numéricos y los caracteres @, _, !, ., ? y $.

Referencias a variables del lenguaje principal en REXX

El intérprete de REXX examina cada serie que no tiene comillas de un procedimiento. Si la serie representa a una variable en la agrupación actual de variables de REXX, éste sustituye la serie por el valor actual. A continuación se muestra un ejemplo de cómo se puede hacer referencia a una variable del lenguaje principal en REXX:

```
CALL SQLEXEC 'FETCH C1 INTO :cm'
SAY 'Commission = ' cm
```

Para asegurarse de que una serie de caracteres no se convierta a un tipo de datos numérico, encierre la serie entre comillas tal como en el ejemplo siguiente:

```
VAR = '100'
```

REXX establece la variable *VAR* en la serie de caracteres de 3 bytes `100`. Si se tienen que incluir comillas como parte de la serie, siga este ejemplo:

```
VAR = "'100'"
```

Cuando se insertan datos numéricos en un campo de tipo CHARACTER, el intérprete de REXX trata los datos numéricos como datos enteros, por lo que deberá concatenar explícitamente las series numéricas y encerrarlas entre comillas.

Variables de indicador en REXX

El tipo de datos de una variable de indicador en REXX es un número sin coma decimal. A continuación se muestra un ejemplo de variable de indicador en REXX que utiliza la palabra clave INDICATOR.

```
CALL SQLEXEC 'FETCH C1 INTO :cm INDICATOR :cmind'
IF ( cmind < 0 )
   SAY 'Commission is NULL'
```

En el ejemplo anterior, se examina que `cmind` tenga un valor negativo. Si no es negativo, la aplicación puede utilizar el valor devuelto de `cm`. Si es negativo, el valor captado es NULL y no se debe utilizar `cm`. En este caso, el gestor de bases de datos no cambia el valor de la variable del lenguaje principal.

Variables de REXX predefinidas

SQLEXEC, SQLDBS y SQLDB2 establecen variables de REXX predefinidas como consecuencia de determinadas operaciones. Estas variables son:

RESULT
Cada operación establece este código de retorno. Los valores posibles son:

n Donde *n* es un valor positivo que indica el número de bytes de un mensaje formateado. La API GET ERROR MESSAGE sola devuelve este valor.

0 Se ha ejecutado la API. La SQLCA de la variable de REXX contiene el estado de terminación de la API. Si SQLCA.SQLCODE es distinto de cero, SQLMSG contiene el mensaje de texto asociado a este valor.

–1 No se dispone de suficiente memoria para completar la API. No se ha devuelto el mensaje solicitado.

–2 SQLCA.SQLCODE se establece en 0. No se ha devuelto ningún mensaje.

–3 SQLCA.SQLCODE contenía un SQLCODE que no era válido. No se ha devuelto ningún mensaje.

–6 No se ha podido crear la variable SQLCA de REXX. Esto indica que no había suficiente memoria disponible o que la agrupación de variables de REXX no estaba disponible por algún motivo.

–7 No se ha podido crear la variable SQLMSG de REXX. Esto indica que no había suficiente memoria disponible o que la agrupación de variables de REXX no estaba disponible por algún motivo.

–8 No se ha podido captar la variable SQLCA.SQLCODE de REXX de la agrupación de variables de REXX.

–9 La variable SQLCA.SQLCODE de REXX se ha truncado durante la captación. La longitud máxima para esta variable es de 5 bytes.

–10 La variable SQLCA.SQLCODE de REXX no se ha podido convertir de ASCII a un entero largo válido.

–11 No se ha podido captar la variable SQLCA.SQLERRML de REXX de la agrupación de variables de REXX.

–12 La variable SQLCA.SQLERRML de REXX se ha truncado durante la captación. La longitud máxima para esta variable es de 2 bytes.

–13 La variable SQLCA.SQLERRML de REXX no se ha podido convertir de ASCII a un entero corto válido.

–14 No se ha podido captar la variable SQLCA.SQLERRMC de REXX de la agrupación de variables de REXX.

–15 La variable SQLCA.SQLERRMC de REXX se ha truncado durante la captación. La longitud máxima para esta variable es de 70 bytes.

–16 No se ha podido establecer la variable de REXX especificada para el texto del error.

–17 No se ha podido captar la variable SQLCA.SQLSTATE de REXX de la agrupación de variables de REXX.

–18 La variable SQLCA.SQLSTATE de REXX se ha truncado durante la captación. La longitud máxima para esta variable es de 2 bytes.

Nota: Los valores –8 a –18 sólo los devuelve la API GET ERROR MESSAGE.

SQLMSG

Si SQLCA.SQLCODE es distinto de cero, esta variable contiene el mensaje de texto asociado al código de error.

SQLISL

Nivel de aislamiento. Los valores posibles son:

RR Lectura repetible.
RS Estabilidad de lectura.
CS Estabilidad del cursor. Éste es el valor por omisión.
UR Lectura no confirmada.
NC Sin confirmación. (NC sólo recibe soporte de algunos servidores de sistema principal, AS/400 o iSeries.)

SQLCA

La estructura SQLCA actualizada después de que se procesen las sentencias de SQL y se llame a las API de DB2.

SQLRODA

La estructura SQLDA de entrada/salida para procedimientos almacenados invocados mediante la sentencia CALL. También es la estructura SQLDA de salida para procedimientos almacenados invocados mediante la API Database Application Remote Interface (DARI).

SQLRIDA

La estructura SQLDA de entrada para procedimientos almacenados invocados mediante la API Database Application Remote Interface (DARI).

SQLRDAT

Una estructura SQLCHAR para procedimientos de servidor invocados mediante la API Database Application Remote Interface (DARI).

Información relacionada:

- "SQLCA" en la publicación *Administrative API Reference*
- "SQLCHAR" en la publicación *Administrative API Reference*
- "SQLDA" en la publicación *Administrative API Reference*

Variables del lenguaje principal de LOB en REXX

Cuando se capte una columna LOB en una variable del lenguaje principal REXX, esta se almacenará como una serie simple (es decir, descontada). Esto se maneja del mismo modo que todos los tipos de SQL basados en caracteres (tales como CHAR, VARCHAR, GRAPHIC, LONG, etc.). En la entrada, si el tamaño del contenido de la variable del lenguaje principal es superior a 32 K, o si cumple con otros criterios establecidos más adelante, se le asignará el tipo de LOB apropiado.

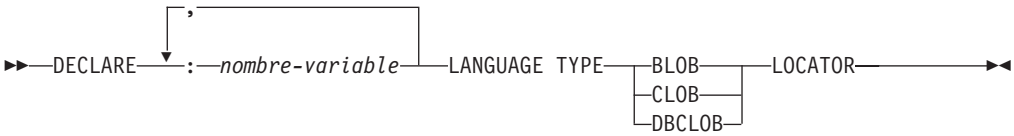
En SQL de REXX, los tipos de SQL se determinan a partir del contenido de la serie de la variable del lenguaje principal, del modo siguiente:

Contenido de la serie de variable del lenguaje principal	Tipo de LOB resultante
:hv1='serie normal entre comillas de más de 32 K ...'	CLOB
:hv2="'serie con comillas de delimitación incorporadas ", "de más de 32 K...'"	CLOB
:hv3="G'serie DBCS con comillas de delimitación incorporadas, ", "que empieza por G, de más de 32 K...'"	DBCLOB
:hv4="BIN'serie con comillas de delimitación incorporadas, ", "que empieza por BIN, de cualquier longitud...'"	BLOB

Sintaxis de las declaraciones de localizador de LOB en REXX

A continuación se muestra la sintaxis para declarar variables del lenguaje principal de localizador de LOB en REXX:

Sintaxis de las variables del lenguaje principal de localizador de LOB en REXX



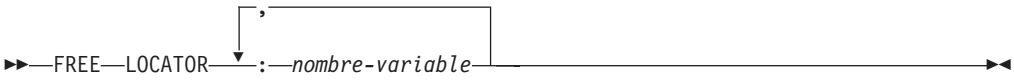
Debe declarar las variables del lenguaje principal de localizador de LOB en la aplicación. Cuando REXX/SQL encuentra estas declaraciones, trata las variables del lenguaje principal declaradas como localizadores durante el resto del programa. Los valores de localizadores se almacenan en variables de REXX, en un formato interno.

Ejemplo:

```
CALL SQLEXEC 'DECLARE :hv1, :hv2 LANGUAGE TYPE CLOB LOCATOR'
```

Los datos representados por localizadores de LOB devueltos por el mecanismo se pueden liberar en REXX/SQL mediante la sentencia FREE LOCATOR, que tiene el formato siguiente:

Sintaxis de la sentencia FREE LOCATOR



Ejemplo:

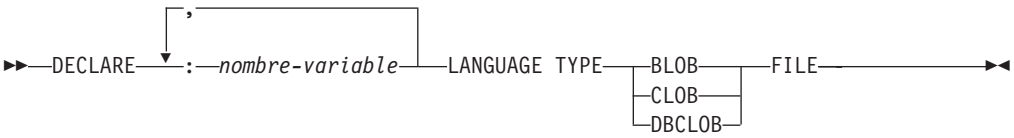
```
CALL SQLEXEC 'FREE LOCATOR :hv1, :hv2'
```

Sintaxis de las declaraciones de referencia de archivos LOB en REXX

Debe declarar las variables del lenguaje principal de referencia de archivos LOB en la aplicación. Cuando REXX/SQL encuentra estas declaraciones, trata las variables del lenguaje principal declaradas como referencias de archivos LOB durante el resto del programa.

A continuación se muestra la sintaxis para declarar variables del lenguaje principal de referencia de archivos LOB en REXX:

Declaraciones de referencia de archivos en REXX



Ejemplo:

```
CALL SQLEXEC 'DECLARE :hv3, :hv4 LANGUAGE TYPE CLOB FILE'
```

Las variables de referencia de archivos en REXX contienen tres campos. Para el ejemplo anterior, éstos son:

hv3.FILE_OPTIONS.
Establecido por la aplicación para indicar cómo se utilizará el archivo.

hv3.DATA_LENGTH.
Establecido por DB2 para indicar el tamaño del archivo.

hv3.NAME.
Establecido por la aplicación con el nombre del archivo LOB.

Para FILE_OPTIONS, la aplicación establece las palabras clave siguientes:

Palabra clave (valor entero)
Significado

READ (2) El archivo se va a utilizar como entrada. Se trata de un archivo normal que se puede abrir, leer y cerrar. La longitud de los datos del archivo (en bytes) se calcula (lo hace el código solicitante de la aplicación) al abrir el archivo.

CREATE (8) En la salida, crear un nuevo archivo. Si el archivo ya existe, es un error. La longitud (en bytes) del archivo se devuelve en el campo DATA_LENGTH de la estructura de la variable de referencia de archivos.

OVERWRITE (16)
En la salida, se sobregraba el archivo, si ya existe; de lo contrario, se crea un nuevo archivo. La longitud (en bytes) del archivo se devuelve en el campo DATA_LENGTH de la estructura de la variable de referencia de archivos.

APPEND (32) La salida se añade al archivo, si existe; de lo contrario, se crea un nuevo archivo. La longitud (en bytes) de los datos que se han añadido al archivo (y no la longitud total del archivo) se devuelve en el campo DATA_LENGTH de la estructura de la variable de referencia de archivos.

Nota: Una variable del lenguaje principal de referencia de archivos es una variable compuesta en REXX, por lo que debe establecer valores para los campos NAME, NAME_LENGTH y FILE_OPTIONS además de declararlas.

Borrado de variables del lenguaje principal de LOB en REXX

En plataformas basadas en Windows®, puede ser necesario borrar explícitamente las declaraciones de variables dl lenguaje principal de referencia de archivos y de localizador de LOB de SQL de REXX, puesto que permanecen en vigor una vez que finaliza el programa de aplicación. Esto es debido a que el proceso de la

aplicación no sale hasta que se cierra la sesión en la que se ejecuta. Si no se borran las declaraciones de LOB del SQL para REXX, pueden interferir con otras aplicaciones que se ejecuten en la misma sesión después de que se haya ejecutado una aplicación de LOB.

La sintaxis para borrar la declaración es:

```
CALL SQLEXEC "CLEAR SQL VARIABLE DECLARATIONS"
```

Debe codificar esta sentencia al final de las aplicaciones de LOB. Observe que la puede codificar en cualquier lugar, como medida de precaución, para borrar las declaraciones que puedan haber quedado de aplicaciones anteriores (por ejemplo, al principio de una aplicación SQL en REXX).

Cursores en REXX

Cuando se declara un cursor en REXX, el cursor se asocia a una consulta. La consulta se asocia a un nombre de sentencia asignado en la sentencia PREPARE. Cualquier variable del lenguaje principal a la que se haga referencia se representa mediante marcadores de parámetros. El siguiente ejemplo muestra una sentencia DECLARE asociada a una sentencia SELECT dinámica:

```
prep_string = "SELECT TABNAME FROM SYSCAT.TABLES WHERE TABSCHEMA = ?"
CALL SQLEXEC 'PREPARE S1 FROM :prep_string';
CALL SQLEXEC 'DECLARE C1 CURSOR FOR S1';
CALL SQLEXEC 'OPEN C1 USING :schema_name';
```

Información relacionada:

- "Tipos de datos SQL soportados en REXX" en la página 539

Tipos de datos SQL soportados en REXX

Determinados tipos de datos de REXX predefinidos corresponden a tipos de columna de DB2. La tabla siguiente muestra cómo SQLEXEC y SQLDBS interpretan variables de REXX para convertir su contenido en tipos de datos de DB2.

Nota: No existe soporte de variables del lenguaje principal para el tipo de datos DATALINK en ninguno de los lenguajes principales de DB2.

Tabla 80. Tipos de columna de SQL correlacionados con declaraciones de REXX

Tipo de columna SQL[1]	Tipo de datos de REXX	Descripción del tipo de columna SQL
SMALLINT (500 ó 501)	Un número sin coma decimal que va de -32 768 a 32 767	Entero con signo de 16 bits
INTEGER (496 ó 497)	Un número sin coma decimal que va de -2 147 483 648 a 2 147 483 647	Entero con signo de 32 bits
REAL[2] (480 ó 481)	Un número en notación científica que va de -3.40282346 x 10^{38} a 3.40282346 x 10^{38}	Coma flotante de precisión única
DOUBLE[3] (480 ó 481)	Un número en notación científica que va de -1.79769313 x 10^{308} a 1.79769313 x 10^{308}	Coma flotante de precisión doble
DECIMAL(*p*,*s*)(484 ó 485)	Un número con una coma decimal	Decimal empaquetado

Tabla 80. Tipos de columna de SQL correlacionados con declaraciones de REXX (continuación)

Tipo de columna SQL[1]	Tipo de datos de REXX	Descripción del tipo de columna SQL
CHAR(*n*) (452 ó 453)	Una serie con una comilla inicial y otra final ('), cuya longitud es *n* después de eliminar las dos comillas Una serie de longitud *n* sin ningún carácter no numérico, aparte de los blancos inicial y final o la E en notación científica	Serie de caracteres de longitud fija con longitud *n*, donde *n* va de 1 a 254
VARCHAR(*n*) (448 ó 449)	Equivalente a CHAR(*n*)	Serie de caracteres de longitud variable con longitud *n*, donde *n* va de 1 a 4000
LONG VARCHAR (456 ó 457)	Equivalente a CHAR(*n*)	Serie de caracteres de longitud variable con longitud *n*, donde *n* va de 1 a 32 700
CLOB(*n*) (408 ó 409)	Equivalente a CHAR(*n*)	Serie de caracteres de longitud variable y de objeto grande con longitud *n*, donde *n* va de 1 a 2 147 483 647
Variable de localizador CLOB[4] (964 ó 965)	DECLARE *:nombre_var* LANGUAGE TYPE CLOB LOCATOR	Identifica las entidades CLOB que residen en el servidor
Variable de referencia de archivo CLOB[4] (920 ó 921)	DECLARE *:nombre_var* LANGUAGE TYPE CLOB FILE	Descriptor de archivo que contiene datos CLOB
BLOB(*n*) (404 ó 405)	Una serie con un apóstrofo inicial y uno final, precedida de BIN, que contiene *n* caracteres después de eliminar el BIN precedente y los dos apóstrofos.	Serie binaria de longitud variable y de objeto grande con longitud *n*, donde *n* va de 1 a 2 147 483 647
Variable de localizador BLOB[4] (960 ó 961)	DECLARE *:nombre_var* LANGUAGE TYPE BLOB LOCATOR	Identifica las entidades BLOB del servidor
Variable de referencia de archivo BLOB[4] (916 ó 917)	DECLARE *:nombre_var* LANGUAGE TYPE BLOB FILE	Descriptor del archivo que contiene datos BLOB
DATE (384 ó 385)	Equivalente a CHAR(*10*)	Serie de caracteres de 10 bytes
TIME (388 ó 389)	Equivalente a CHAR(*8*)	Serie de caracteres de 8 bytes
TIMESTAMP (392 ó 393)	Equivalente a CHAR(*26*)	Serie de caracteres de 26 bytes
Nota: Los tipos de datos siguientes sólo están disponibles en el entorno DBCS.		
GRAPHIC(*n*) (468 ó 469)	Una serie con un apóstrofo inicial y uno final, precedida de una G o una N, que contiene *n* caracteres DBCS después de eliminar el carácter precedente y los dos apóstrofos	Serie gráfica de longitud fija con longitud *n*, donde *n* va de 1 a 127
VARGRAPHIC(*n*) (464 ó 465)	Equivalente a GRAPHIC(*n*)	Serie gráfica de longitud variable con longitud *n*, donde *n* va de 1 a 2000
LONG VARGRAPHIC (472 ó 473)	Equivalente a GRAPHIC(*n*)	Serie gráfica de longitud variable larga con longitud *n*, donde *n* va de 1 a 16 350
DBCLOB(*n*) (412 ó 413)	Equivalente a GRAPHIC(*n*)	Serie gráfica de longitud variable y de objeto grande con longitud *n*, donde *n* va de 1 a 1 073 741 823
Variable de localizador DBCLOB[4] (968 ó 969)	DECLARE *:nombre_var* LANGUAGE TYPE DBCLOB LOCATOR	Identifica las entidades DBCLOB que residen en el servidor
Variable de referencia de archivos DBCLOB[4] (924 ó 925)	DECLARE *:nombre_var* LANGUAGE TYPE DBCLOB FILE	Descriptor de archivo que contiene datos DBCLOB

Tabla 80. Tipos de columna de SQL correlacionados con declaraciones de REXX (continuación)

Tipo de columna SQL[1]	Tipo de datos de REXX	Descripción del tipo de columna SQL

Notas:

1. El primer número que se encuentra bajo **Tipo de columna SQL** indica que no se proporciona una variable de indicador, y el segundo número indica que se proporciona una variable de indicador. Se necesita una variable de indicador para indicar los valores nulos (NULL) o para contener la longitud de una serie truncada.
2. FLOAT(*n*) donde 0 < *n* < 25 es un sinónimo de REAL. La diferencia entre REAL y DOUBLE en la SQLDA es el valor de la longitud (4 u 8).
3. Los tipos SQL siguientes son sinónimos de DOUBLE:
   - FLOAT
   - FLOAT(*n*) donde 24 < *n* < 54 es
   - DOUBLE PRECISION
4. Éste no es un tipo de columna, sino un tipo de variable del lenguaje principal.

Conceptos relacionados:

- "Cursores en REXX" en la página 539

Requisitos de ejecución para REXX

Las siguientes secciones describen los requisitos de ejecución para aplicaciones REXX.

Creación y ejecución de aplicaciones REXX

Las aplicaciones REXX no se precompilan, compilan ni enlazan. Las instrucciones siguientes describen cómo crear y ejecutar aplicaciones REXX en sistemas operativos Windows y en el sistema operativo AIX.

Restricciones:

En plataformas basadas en Windows, el archivo de aplicación debe tener una extensión .CMD. Después de su creación, puede ejecutar la aplicación directamente desde el indicador de mandatos del sistema operativo. En AIX, el archivo de la aplicación puede tener cualquier extensión.

Procedimiento:

Cree y ejecute las aplicaciones REXX del siguiente modo:

- En sistemas operativos Windows, el archivo de aplicación puede tener cualquier nombre. Después de su creación, puede ejecutar la aplicación desde el indicador de mandatos del sistema operativo invocando al intérprete de REXX del modo siguiente:

  ```
  REXX file_name
  ```

- En AIX, puede ejecutar la aplicación mediante cualquiera de los dos métodos siguientes:
  - En el indicador de mandatos de shell, escriba `rexx name` donde `name` es el nombre del programa REXX.
  - Si la primera línea del programa REXX contiene un "número mágico" (#!) e identifica el directorio en que reside el intérprete de REXX/6000, puede ejecutar el programa REXX escribiendo su nombre en el indicador de mandatos del shell. Por ejemplo, si el archivo del intérprete de REXX/6000 está en el directorio /usr/bin, incluya la información siguiente como primera línea del programa REXX:

    ```
    #! /usr/bin/rexx
    ```

A continuación, haga que el programa sea ejecutable escribiendo el mandato siguiente en el indicador de mandatos del shell:

```
chmod +x name
```

Ejecute el programa REXX escribiendo su nombre de archivo en el indicador de mandatos del shell.

Nota: En AIX, debe establecer la variable de entorno LIBPATH para incluir el directorio en que está ubicada la biblioteca de SQL para REXX, `db2rexx`. Por ejemplo:

```
export LIBPATH=/lib:/usr/lib:/usr/lpp/db2_08_01/lib
```

Archivos de vinculación de REXX

Se proporcionan cinco archivos de vinculación para el soporte de aplicaciones REXX. Los nombres de estos archivos están incluidos en el archivo DB2UBIND.LST. Cada archivo de vinculación se ha precompilado utilizando un nivel de aislamiento distinto; por consiguiente, en la base de datos hay cinco paquetes diferentes almacenados.

Los cinco archivos de vinculación son:

DB2ARXCS.BND
Soporta el nivel de aislamiento de estabilidad del cursor.

DB2ARXRR.BND
Soporta el nivel de aislamiento de estabilidad de lectura repetible.

DB2ARXUR.BND
Soporta el nivel de aislamiento de estabilidad de lectura no confirmada.

DB2ARXRS.BND
Soporta el nivel de aislamiento de estabilidad de lectura.

DB2ARXNC.BND
Soporta el nivel de aislamiento de sin confirmación. Se utiliza este nivel de aislamiento cuando se trabaja con algunos servidores de bases de datos de sistema principal, AS/400 o iSeries. En otras bases de datos, se comporta como el nivel de aislamiento de lectura no confirmada.

Nota: En algunos casos, puede que sea necesario vincular de forma explícita estos archivos a la base de datos.

Cuando se utiliza la rutina SQLEXEC se usa, por omisión, el paquete creado con estabilidad del cursor. Si necesita uno de los otros niveles de aislamiento, lo puede cambiar mediante la API SQLDBS CHANGE SQL ISOLATION LEVEL antes de conectar con la base de datos. Esto ocasionará que las posteriores llamadas a la rutina SQLEXEC se asocien al nivel de aislamiento especificado.

Las aplicaciones REXX basadas en Windows no pueden asumir que esté en vigor el nivel de aislamiento por omisión a menos que sepan que ningún otro programa REXX de la sesión ha cambiado el valor. Antes de conectar con una base de datos, una aplicación REXX debe establecer explícitamente el nivel de aislamiento.

Sintaxis de las API para REXX

Utilice la rutina SQLDBS para llamar a las API de DB2 con la sintaxis siguiente:

```
CALL SQLDBS 'command string'
```

Si no puede llamar a una API de DB2® que desea utilizar mediante la rutina SQLDBS, puede llamar a la API efectuando una llamada al procesador de línea de mandatos (CLP) de DB2 desde dentro de la aplicación REXX. Sin embargo, puesto que el CLP de DB2 dirige la salida al dispositivo de salida estándar o a un archivo especificado, la aplicación REXX no puede acceder directamente a la salida de la API de DB2 a la que se ha llamado, ni puede tomar fácilmente la determinación de si la API llamada ha resultado satisfactoria o no. La API SQLDB2 proporciona una interfaz con el CLP de DB2 que proporciona información directa a la aplicación REXX sobre el éxito o fracaso de cada API llamada, estableciendo la variable compuesta SQLCA de REXX después de cada llamada.

Puede utilizar la rutina SQLDB2 para llamar a las API de DB2 utilizando la sintaxis siguiente:

```
CALL SQLDB2 'command string'
```

donde `'command string'` es una serie que puede procesar el procesador de línea de mandatos (CLP).

El hecho de llamar a una API de DB2 mediante SQLDB2 es equivalente a llamar directamente al CLP, excepto en los aspectos siguientes:

- La llamada al ejecutable del CLP se sustituye por la llamada a SQLDB2 (el resto de opciones y parámetros del CLP se especifican de la misma manera).
- La variable compuesta SQLCA de REXX se establece después de llamar a SQLDB2, pero no se establece después de llamar al ejecutable del CLP.
- La salida de visualización por omisión del CLP se establece como desactivada cuando se llama a SQLDB2, mientras que se establece como salida activada cuando se llama al ejecutable del CLP. Observe que puede activar la salida de visualización del CLP pasando las opciones `+o` o `-o-` a SQLDB2.

Puesto que la única variable de REXX que se establece después de llamar a SQLDB2 es la SQLCA, sólo debe utilizar esta rutina para llamar a las API de DB2 que no devuelvan datos distintos de la SQLCA y que no estén implementadas actualmente mediante la interfaz SQLDBS. Así pues, SQLDB2 únicamente soporta las API de DB2 siguientes:

- Activar base de datos
- Añadir nodo
- Vincular para DB2 Versión 1(1) (2)
- Vincular para DB2 Versión 2 ó 5(1)
- Crear base de datos en nodo
- Eliminar base de datos en nodo
- Verificar eliminación de nodo
- Desactivar base de datos
- Desregistrar
- Cargar(3)
- Cargar consulta
- Precompilar programa(1)
- Revincular paquete(1)
- Redistribuir grupo de particiones de base de datos
- Registrar
- Iniciar gestor de la base de datos
- Detener gestor de la base de datos

Notas sobre las API de DB2 soportadas por SQLDB2:

1. Estos mandatos requieren una sentencia CONNECT a través de la interfaz SQLDB2. Las conexiones que utilizan la interfaz SQLDB2 no están accesibles

para la interfaz SQLEXEC, y las conexiones que utilizan la interfaz SQLEXEC no están accesible para la interfaz SQLDB2.

2. Está soportado en las plataformas basadas en Windows® mediante la interfaz SQLDB2.
3. El parámetro de salida opcional, `pLoadInfoOut` para la API Cargar no se devuelve a la aplicación en REXX.

Nota: Aunque la rutina SQLDB2 ha sido pensada para uso únicamente de las API de DB2 relacionadas anteriormente, también se puede utilizar para otras API de DB2 que no se soportan a través de la rutina SQLDBS. Alternativamente, se puede acceder a las API de DB2 a través del CLP desde la aplicación REXX.

Llamada a procedimientos almacenados desde REXX

Las secciones siguientes describen cómo llamar a procedimientos almacenados desde aplicaciones REXX.

Procedimientos almacenados en REXX

Las aplicaciones SQL de REXX pueden llamar a procedimientos almacenados que se encuentren en el servidor de bases de datos utilizando la sentencia CALL de SQL. El procedimiento almacenado puede estar escrito en cualquier lenguaje soportado en dicho servidor, a excepción de REXX en los sistemas AIX®. (Las aplicaciones cliente pueden estar escritas en REXX en los sistemas AIX, pero, al igual que en otros lenguajes, no pueden llamar a un procedimiento almacenado escrito en REXX en AIX.)

Conceptos relacionados:

• "Llamadas a procedimientos almacenados en REXX" en la página 544

Llamadas a procedimientos almacenados en REXX

La sentencia CALL permite que una aplicación cliente pase datos a un procedimiento almacenado en un servidor, y reciba datos del mismo. La interfaz para los datos tanto de entrada como de salida es una lista de variables del lenguaje principal. Puesto que REXX suele determinar el tipo y el tamaño de las variables del lenguaje principal en base a su contenido, las variables que sean sólo de salida y se pasen a CALL se deben inicializar con datos *ficticios*, parecidos en tipo y tamaño a la salida esperada.

También se pueden pasar datos a los procedimientos almacenados a través de las variables SQLDA de REXX, mediante la sintaxis USING DESCRIPTOR de la sentencia CALL. La tabla siguiente muestra cómo se establece la SQLDA. En la tabla, ':value' es la raíz de una variable del lenguaje principal REXX que contiene los valores necesarios para la aplicación. Para DESCRIPTOR, 'n' es un valor numérico que indica un elemento *sqlvar* específico de la SQLDA. Los números que aparecen a la derecha hacen referencia a las notas que siguen a la tabla.

Tabla 81. SQLDA de REXX de la parte cliente para procedimientos almacenados que utilizan la sentencia CALL

USING DESCRIPTOR	:value.SQLD	1
	:value.n.SQLTYPE	1
	:value.n.SQLLEN	1

Tabla 81. SQLDA de REXX de la parte cliente para procedimientos almacenados que utilizan la sentencia CALL (continuación)

	:value.n.SQLDATA	1	2
	:value.n.SQLDIND	1	2

Notas:

1. Antes de invocar al procedimiento almacenado, la aplicación cliente tiene que inicializar la variable de REXX con los datos apropiados.

   Cuando se ejecuta la sentencia CALL de SQL, el gestor de bases de datos asigna almacenamiento y recupera el valor de la variable de REXX de la agrupación de variables de REXX. Para una SQLDA utilizada en una sentencia CALL, el gestor de bases de datos asigna almacenamiento para los campos SQLDATA y SQLIND en base a los valores de SQLTYPE y SQLLEN.

   En el caso de un procedimiento almacenado REXX (es decir, el propio procedimiento invocado está escrito en REXX basado en Windows®), los datos pasados por el cliente desde cualquiera de los dos tipos de sentencia CALL o de la API DARI se colocan en la agrupación de variables de REXX en el servidor de bases de datos, utilizando los nombres predefinidos siguientes:

   SQLRIDA
   Nombre predefinido para la variable SQLDA de entrada de REXX

   SQLRODA
   Nombre predefinido para la variable SQLDA de salida de REXX
2. Cuando termina el procedimiento almacenado, el gestor de bases de datos recupera también el valor de las variables del procedimiento almacenado. Los valores se devuelven a la aplicación cliente y se colocan en la agrupación de variables de REXX del cliente.

Conceptos relacionados:

- "Consideraciones del cliente para llamar a procedimientos almacenados en REXX" en la página 545
- "Consideraciones del servidor para llamar a procedimientos almacenados en REXX" en la página 546
- "Recuperación de valores de precisión y escala (SCALE) de campos decimales de la SQLDA" en la página 546

Información relacionada:

- "Sentencia CALL" en la publicación *Consulta de SQL, Volumen 2*

Consideraciones del cliente para llamar a procedimientos almacenados en REXX

Cuando utilice variables del lenguaje principal en la sentencia CALL, inicialice cada una de las variables del lenguaje principal con un valor que sea de un tipo compatible con los datos que se devuelvan a la variable del lenguaje principal desde el procedimiento del servidor. Debe realizar esta inicialización aunque el indicador correspondiente sea negativo.

Cuando utilice descriptores, SQLDATA debe estar inicializada y contener datos que sean de un tipo compatible con los datos que se devuelvan desde el procedimiento del servidor. Debe realizar esta inicialización aunque el campo SQLIND contenga un valor negativo.

Información relacionada:

- "Tipos de datos SQL soportados en REXX" en la página 539

Consideraciones del servidor para llamar a procedimientos almacenados en REXX

Asegúrese de que todos los campos SQLDATA y SQLIND (si es de un tipo anulable) de la SQLRODA de SQLDA de salida predefinida estén inicializados. Por ejemplo, si SQLRODA.SQLD es 2, los campos siguientes deben contener datos (aunque los indicadores correspondientes sean negativos y no se pasen datos al cliente):

- SQLRODA.1.SQLDATA
- SQLRODA.2.SQLDATA

Recuperación de valores de precisión y escala (SCALE) de campos decimales de la SQLDA

Para recuperar los valores de precisión y escala para campos decimales de la estructura SQLDA devuelta por el gestor de bases de datos, utilice los valores `sqllen.scale` y `sqllen.precision` cuando inicialice la salida de la SQLDA en el programa REXX. Por ejemplo:

```
  .
  .
  .
/* INICIALIZAR UN ELEMENTO DE SQLDA DE SALIDA */
io_sqlda.sqld = 1
io_sqlda.1.sqltype = 485           /* TIPO DE DATOS DECIMAL */
io_sqlda.1.sqllen.scale  = 2       /* DÍGITOS A LA DERECHA DE LA COMA DECIMAL */
io_sqlda.1.sqllen.precision  = 7   /* ANCHURA DEL DECIMAL  */
io_sqlda.1.sqldata = 00000.00      /* FORMATO DE DATOS DE DEFINICIÓN DE AYUDAS */
io_sqlda.1.sqlind = -1             /* SIN DATOS DE ENTRADA */
  .
  .
  .
```

Capítulo 24. Escritura de aplicaciones utilizando funciones de DB2 WebSphere MQ

Visión general funcional de WebSphere MQ . . . 547
Mensajería de WebSphere MQ. 549
Envío de mensajes con funciones de WebSphere MQ. 552
Recuperación de mensajes con funciones de WebSphere MQ. 553
Conectividad de aplicación a aplicación de WebSphere MQ. 555
Comunicaciones de solicitud/respuesta con funciones de WebSphere MQ 557
Publicación/suscripción con funciones de WebSphere MQ. 558

Visión general funcional de WebSphere MQ

WebSphere® MQ proporciona la capacidad de combinar operaciones de mensajes y operaciones de bases de datos, y en algunos casos en una sola unidad de trabajo, para crear una transacción atómica. Esta función recibe soporte de las funciones de WebSphere MQ en UNIX® y Windows® con funciones transacciones y no transaccionales definidas por el usuario de MQ, mediante los esquemas DB2MQ y DB2MQ1C.

Se proporciona un conjunto de funciones de WebSphere MQ con DB2® para permitir que las sentencias de SQL incluyan operaciones de mensajería. Esto significa que este soporte está disponible para aplicaciones escritas en cualquier lenguaje soportado, como por ejemplo C, JavaT™, SQL mediante cualquiera de las interfaces de bases de datos. Todos los ejemplos que se muestran a continuación están en SQL. Este SQL se puede utilizar desde otros lenguajes de programación en todas las formas habituales. Están soportados todos los estilos de mensajería de WebSphere MQ descritos anteriormente.

En una configuración básica, un servidor WebSphere MQ está situado en la máquina servidor de bases de datos junto con DB2. Las funciones de WebSphere MQ se instalan en DB2 y proporcionan acceso al servidor WebSphere MQ. Los clientes DB2 pueden estar situados en cualquier máquina a la que pueda acceder el servidor DB2. Varios clientes pueden acceder simultáneamente a funciones de WebSphere MQ a través de la base de datos. A través de las funciones proporcionadas, los clientes DB2 pueden realizar funciones de mensajería dentro de sentencias de SQL. Estas operaciones de mensajería permiten que aplicaciones DB2 se comuniquen entre ellas o con otras aplicaciones WebSphere MQ.

El mandato **enable_MQFunctions** sirve para habilitar una base de datos DB2 para las funciones de WebSphere MQ. Establecerá automáticamente una configuración por omisión sencilla que pueden utilizar las aplicaciones cliente sin más acción administrativa. Para ver una descripción, consulte los temas ″enable_MQFunctions″ y ″disable_MQFunctions″. La configuración por omisión ofrece a los programadores de aplicaciones una forma rápida de comenzar a utilizar el producto y una interfaz sencilla para el desarrollo. Se pueden configurar funciones adicionales de forma incremental a medida que se necesitan.

Para enviar un mensaje sencillo utilizando la configuración por omisión, utilice la siguiente sentencia de SQL:

```
Ejemplo 1:
VALUES DB2MQ.MQSEND('simple message')
```

Esto envía el mensaje “simple message” al gestor de colas de WebSphere MQ y a la cola especificada por la configuración por omisión.

La interfaz denominada Application Messaging Interface (AMI) de WebSphere MQ proporciona una clara separación entre acciones de mensajería y las definiciones y controla el modo en que se deben llevar a cabo estas acciones. Estas funciones se mantienen un archivo de depósito externo y se gestionan mediante la herramienta de administración de AMI. Esto hace que las aplicaciones AMI sean fáciles de desarrollar y de mantener. Las funciones de WebSphere MQ que se proporcionan con DB2 Universal Database™ se basan en la interfaz AMI de WebSphere MQ. AMI da soporte al uso de un archivo de configuración externo, denominado depósito de AMI, para almacenar información de configuración. La configuración por omisión incluye un depósito de AMI de WebSphere MQ configurado para su uso con DB2 Universal Database.

Dos conceptos clave de WebSphere MQ AMI, puntos de servicio y políticas, se utilizan en las funciones de DB2 WebSphere MQ. Un punto de servicio es un punto final lógico desde el que se puede enviar o recibir un mensaje. En el depósito de AMI, cada punto de servicio está definido con un nombre de cola de WebSphere MQ y un gestor de colas. Las políticas definen la calidad de las opciones de servicio que se debe utilizar para una determinada operación de mensajería. Las calidades clave de servicio incluyen prioridad y permanencia del mensaje. Se proporcional puntos de servicio por omisión y definiciones de políticas que pueden utilizar los desarrolladores para simplificar sus aplicaciones. El ejemplo de “Ejemplo 1” en la página 547 se puede reescribir del siguiente modo para especificar de forma explícita el punto de servicio y el nombre de política por omisión:

```
Ejemplo 2:
VALUES DB2MQ.MQSEND('DB2.DEFAULT.SERVICE',
                    'DB2.DEFAULT.POLICY',
                    'simple message')
```

Las colas se pueden suministrar mediante una o más aplicaciones en el servidor en el que residen las colas y las aplicaciones. En muchas configuraciones se definen varias colas para dar soporte a distintas aplicaciones y finalidades. Por este motivo, suele ser importante definir puntos de servicio cuando se realizan solicitudes de WebSphere MQ. Esto se muestra en el siguiente ejemplo:

```
Ejemplo 3:
VALUES DB2MQ.MQSEND('ODS_Input', 'simple message')
```

En el mensaje anterior, la política no está especificada y por lo tanto se utiliza la política por omisión.

Limitaciones

Cuando se utilizan las funciones de envío o recepción, la longitud máxima de un mensaje de tipo VARCHAR es 4.000 caracteres para el esquema DB2MQ y 32.000 caracteres para el esquema DB2MQ1C. La longitud máxima cuando se envía o se recibe un mensaje de tipo CLOB es 1 MB para el esquema DB2MQ. También hay tamaños máximos de mensaje para publicar un mensaje mediante MQPublish.

A veces se necesitan distintas funciones cuando se trabaja con mensajes CLOB y con mensajes VARCHAR. Generalmente, la versión CLOB de una función MQ utiliza una sintaxis idéntica a la de su correspondiente. La única diferencia es que su nombre tiene los caracteres CLOB al final. Por ejemplo, el equivalente CLOB de MQREAD es MQREADCLOB.

Códigos de error

Los códigos de retorno de las funciones de WebSphere MQ se encuentran en el Apéndice B del manual MQSeries® Application Messaging Interface Manual. Otros mensajes, que utilizan el identificador de componente AMI, se muestran en el manual "DB2 Universal Database, Mensajes y códigos".

Conceptos relacionados:

- "Habilitación de MQSeries" en la página 17
- "Mensajería de WebSphere MQ" en la página 549
- "Envío de mensajes con funciones de WebSphere MQ" en la página 552
- "Recuperación de mensajes con funciones de WebSphere MQ" en la página 553
- "Conectividad de aplicación a aplicación de WebSphere MQ" en la página 555
- "Comunicaciones de solicitud/respuesta con funciones de WebSphere MQ" en la página 557
- "Publicación/suscripción con funciones de WebSphere MQ" en la página 558
- "Cómo utilizar funciones de WebSphere MQ en DB2" en la publicación *IBM DB2 Information Integrator Application Developer's Guide*

Tareas relacionadas:

- "Configuración de funciones de DB2 WebSphere MQ" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Información relacionada:

- "Función escalar MQSEND" en las *Rutinas administrativas de SQL*
- "Función escalar MQRECEIVE" en las *Rutinas administrativas de SQL*
- "Función escalar MQREAD" en las *Rutinas administrativas de SQL*
- "Función escalar MQPUBLISH" en las *Rutinas administrativas de SQL*
- "Función escalar MQSUBSCRIBE" en las *Rutinas administrativas de SQL*
- "Función escalar MQUNSUBSCRIBE" en las *Rutinas administrativas de SQL*
- "Función de tabla MQREADALL" en las *Rutinas administrativas de SQL*
- "Función de tabla MQRECEIVEALL" en las *Rutinas administrativas de SQL*
- "Función escalar MQREADCLOB" en las *Rutinas administrativas de SQL*
- "Función escalar MQRECEIVECLOB" en las *Rutinas administrativas de SQL*
- "Función de tabla MQREADALLCLOB" en las *Rutinas administrativas de SQL*
- "Función de tabla MQRECEIVEALLCLOB" en las *Rutinas administrativas de SQL*
- "Mandato db2mqlsn - MQ Listener" en la publicación *Consulta de mandatos*
- "enable_MQFunctions" en la publicación *Consulta de mandatos*
- "disable_MQFunctions" en la publicación *Consulta de mandatos*

Mensajería de WebSphere MQ

Las funciones de DB2® WebSphere® MQ dan soporte a tres modelos de mensajería: datagramas, publicación/suscripción (p/s) y solicitud/respuesta (r/r). Los mensajes que se envían como datagramas se envían a un solo de destino y no se espera respuesta. En el modelo p/s, uno o más publicadores envían un mensaje a un servicio de publicación que distribuye el mensaje a uno o más suscriptores. El modelo de solicitud/respuesta es parecido al de datagrama, pero el remitente espera recibir una respuesta.

El propio WebSphere MQ no obliga a utilizar ninguna estructura particular de los mensajes que transporta. Otros productos, como MQSeries® Integrator (MQSI),

ofrecen soporte para mensajes formados como C o Cobol o como series XML. Los mensajes estructurados en MQSI se definen a través de un depósito de mensajes. Los mensajes XML suelen tener una estructura de mensaje de auto descripción y también se pueden gestionar a través del depósito. Los mensajes también pueden ser no estructurados, en cuyo caso se necesita código del usuario para analizar o construir el contenido del mensaje. Este tipo de mensajes suelen ser semi estructurados, es decir, utilizan posiciones de byte o delimitadores fijos para separar los campos de un mensaje. El soporte de mensajes semi estructurados se proporciona a través del Asistente de ayuda de WebSphere MQ. El soporte de mensajes XML se proporciona a través de funciones de DB2 XML Extender.

El formato más básico de mensajería con las funciones de WebSphere MQ DB2 se produce cuando todas las aplicaciones de bases de datos se conectan al mismo servidor DB2. Los clientes pueden ser locales al servidor de bases de datos o distribuidos en un entorno de red.

En un escenario sencillo, el Cliente A invoca la función MQSEND para enviar una serie definida por el usuario a la ubicación de servicio por omisión. Las funciones de WebSphere MQ se ejecutan dentro de DB2 en el servidor de bases de datos. Más adelante, el Cliente B invoca la función MQRECEIVE para eliminar el mensaje de la cabecera de la cola definida por el servicio por omisión y devolverlo al cliente. De nuevo, DB2 ejecuta las funciones de WebSphere MQ utilizadas para realizar esta tarea.

Los clientes de bases de datos pueden utilizar la mensajería sencilla de varias formas. Algunos de los usos comunes de la mensajería son:

Recopilación de datos
La información se recibe en el formato de mensajes de una o más fuentes de información diversas. Las fuentes de información pueden ser aplicaciones comerciales, como SAP, o aplicaciones desarrolladas en la empresa. Estos datos se pueden recibir de varias colas y se pueden almacenar en tablas de bases de datos para su futuro proceso y análisis.

Distribución de cargas de trabajo
Las solicitudes de trabajo se colocan en una cola que comparten varias instancias de la misma aplicación. Cuando una instancia está lista para realizar algún trabajo, recibe un mensaje de la parte superior de la cola que contiene la solicitud de trabajo que hay que realizar. Mediante esta técnica, varias instancias pueden compartir la carga de trabajo representada por una sola cola de solicitudes agrupadas.

Señalización de aplicaciones
En una situación en la que colaboran varios procesos, se suelen utilizar mensajes para coordinar sus efectos. Estos mensajes pueden contener mandatos o solicitudes de trabajo que hay que llevar a cabo. Generalmente, este tipo de señalización es de una vía, es decir, el partícipe que inicia el mensaje no acepta una respuesta. Consulte el tema ″Comunicaciones de solicitud/respuesta con funciones de WebSphere MQ″ si desea obtener más información.

Notificación de aplicaciones
La notificación es parecida a la señalización en el hecho de que los datos se envían desde un iniciador que no espera una respuesta. Sin embargo, generalmente la notificación contiene datos sobre sucesos comerciales que han tenido lugar. La publicación/suscripción es una forma más avanzada de publicación. Si desea obtener más información, consulte el tema ″Publicación/suscripción con funciones de WebSphere MQ″.

El escenario siguiente amplía el escenario sencillo descrito anteriormente para incorporar mensajería remota. Es decir, se envía un mensaje entre la Máquina A y la Máquina B. La secuencia de pasos es la siguiente:

1. El Cliente DB2 ejecuta una llamada MQSEND, especificando un servicio de destino que se ha definido para representar una cola remota en la Máquina B.
2. Las funciones de WebSphere MQ DB2 realizan el trabajo real de WebSphere MQ para enviar el mensaje. El servidor WebSphere MQ de la Máquina A acepta el mensaje y garantiza que lo distribuirá al destino definido por la definición de punto de servicio y la configuración actual de WebSphere MQ de la Máquina A. El servidor determina que es una cola de la Máquina B. Luego intenta distribuir el mensaje al servidor WebSphere MQ de la Máquina B, reintentando la acción de forma transparente según lo necesario.
3. El servidor WebSphere MQ de la Máquina B acepta el mensaje procedente del servidor de la Máquina A y lo coloca en la cola de destino de la Máquina B.
4. Un cliente WebSphere MQ de la Máquina B solicita el mensaje de la cabecera de la cola.

Conceptos relacionados:

- "Habilitación de MQSeries" en la página 17
- "Visión general funcional de WebSphere MQ" en la página 547
- "Envío de mensajes con funciones de WebSphere MQ" en la página 552
- "Recuperación de mensajes con funciones de WebSphere MQ" en la página 553
- "Conectividad de aplicación a aplicación de WebSphere MQ" en la página 555
- "Comunicaciones de solicitud/respuesta con funciones de WebSphere MQ" en la página 557
- "Publicación/suscripción con funciones de WebSphere MQ" en la página 558
- "Cómo utilizar funciones de WebSphere MQ en DB2" en la publicación *IBM DB2 Information Integrator Application Developer's Guide*

Tareas relacionadas:

- "Configuración de funciones de DB2 WebSphere MQ" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Información relacionada:

- "Función escalar MQSEND" en las *Rutinas administrativas de SQL*
- "Función escalar MQRECEIVE" en las *Rutinas administrativas de SQL*
- "Función escalar MQREAD" en las *Rutinas administrativas de SQL*
- "Función escalar MQPUBLISH" en las *Rutinas administrativas de SQL*
- "Función escalar MQSUBSCRIBE" en las *Rutinas administrativas de SQL*
- "Función escalar MQUNSUBSCRIBE" en las *Rutinas administrativas de SQL*
- "Función de tabla MQREADALL" en las *Rutinas administrativas de SQL*
- "Función de tabla MQRECEIVEALL" en las *Rutinas administrativas de SQL*
- "Función escalar MQREADCLOB" en las *Rutinas administrativas de SQL*
- "Función escalar MQRECEIVECLOB" en las *Rutinas administrativas de SQL*
- "Función de tabla MQREADALLCLOB" en las *Rutinas administrativas de SQL*
- "Función de tabla MQRECEIVEALLCLOB" en las *Rutinas administrativas de SQL*
- "Mandato db2mqlsn - MQ Listener" en la publicación *Consulta de mandatos*
- "enable_MQFunctions" en la publicación *Consulta de mandatos*
- "disable_MQFunctions" en la publicación *Consulta de mandatos*

Envío de mensajes con funciones de WebSphere MQ

Mediante MQSEND, un usuario de DB2® elige los datos que desea enviar, dónde enviarlos y cuándo se enviarán. En la industria esto se suele denominar "Enviar y olvidar", lo que significa que el remitente simplemente envía el mensaje y confía en los protocolos de distribución garantizados de WebSphere® MQ para asegurar que el mensaje llega a su destino. Los siguientes ejemplos lo ilustran.

El "Ejemplo 4" envía una serie definida por el usuario al punto de servicio myPlace con la política highPriority:

```
Ejemplo 4:
VALUES DB2MQ.MQSEND('myplace','highPriority','test')
```

En el "Ejemplo 4", la política highPriority hace referencia a una política definida en el depósito de AMI que establece la prioridad de WebSphere MQ en el nivel más alto y quizás también ajusta otras calidades de servicio, como la permanencia.

El contenido del mensaje se puede ser cualquier combinación válida de datos de SQL y especificados por el usuario. Esto incluye funciones anidadas, operadores y conversiones de tipos de datos. Por ejemplo, dada una tabla EMPLOYEE, con columnas VARCHAR LASTNAME, FIRSTNAME y DEPARTMENT, si desea enviar un mensaje con esta información correspondiente a cada empleado en DEPARTMENT 5LGA, haría lo siguiente:

```
Ejemplo 5:
SELECT DB2MQ.MQSEND(LASTNAME || ' ' || FIRSTNAME || ' ' || DEPARTMENT)
  FROM EMPLOYEE WHERE DEPARTMENT = '5LGA'
```

Si esta tabla también tuviera una columna AGE definida como un entero, podría incluirla añadiendo la siguiente sentencia:

```
Ejemplo 6:
SELECT DB2MQ.MQSEND (LASTNAME || ' ' || FIRSTNAME || ' ' || DEPARTMENT ||
  ' ' || char(AGE)) FROM EMPLOYEE WHERE DEPARTMENT = '5LGA'
```

Si la tabla EMPLOYEE tuviera una columna RESUME de tipo CLOB en lugar de una columna AGE, se podría emitir un mensaje con información correspondiente a cada empleado en DEPARTMENT 5LGA con la siguiente sentencia:

```
Ejemplo 7:
SELECT DB2MQ.MQSEND(clob(LASTNAME) || ' ' || clob(FIRSTNAME) ||
  ' ' || clob(DEPARTMENT) || ' ' || RESUME))
  FROM EMPLOYEE WHERE DEPARTMENT = '5LGA'
```

El siguiente ejemplo muestra cómo se puede obtener contenido del mensaje utilizando una expresión SQL válida. Dada una segunda tabla DEPT que contiene columnas VARCHAR DEPT_NO y DEPT_NAME, se pueden enviar mensajes que contengan los atributos LASTNAME y DEPT_NAME de cada empleado:

```
Ejemplo 8:
SELECT DB2MQ.MQSEND(e.LASTNAME || ' ' || d.DEPTNAME)
  FROM EMPLOYEE e, DEPT d
  WHERE e.DEPARTMENT = d.DEPTNAME
```

Conceptos relacionados:

- "Habilitación de MQSeries" en la página 17
- "Visión general funcional de WebSphere MQ" en la página 547
- "Mensajería de WebSphere MQ" en la página 549
- "Recuperación de mensajes con funciones de WebSphere MQ" en la página 553
- "Conectividad de aplicación a aplicación de WebSphere MQ" en la página 555

- "Comunicaciones de solicitud/respuesta con funciones de WebSphere MQ" en la página 557
- "Publicación/suscripción con funciones de WebSphere MQ" en la página 558
- "Cómo utilizar funciones de WebSphere MQ en DB2" en la publicación *IBM DB2 Information Integrator Application Developer's Guide*

Tareas relacionadas:

- "Configuración de funciones de DB2 WebSphere MQ" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Información relacionada:

- "Función escalar MQSEND" en las *Rutinas administrativas de SQL*
- "Función escalar MQRECEIVE" en las *Rutinas administrativas de SQL*
- "Función escalar MQREAD" en las *Rutinas administrativas de SQL*
- "Función escalar MQPUBLISH" en las *Rutinas administrativas de SQL*
- "Función escalar MQSUBSCRIBE" en las *Rutinas administrativas de SQL*
- "Función escalar MQUNSUBSCRIBE" en las *Rutinas administrativas de SQL*
- "Función de tabla MQREADALL" en las *Rutinas administrativas de SQL*
- "Función de tabla MQRECEIVEALL" en las *Rutinas administrativas de SQL*
- "Función escalar MQREADCLOB" en las *Rutinas administrativas de SQL*
- "Función escalar MQRECEIVECLOB" en las *Rutinas administrativas de SQL*
- "Función de tabla MQREADALLCLOB" en las *Rutinas administrativas de SQL*
- "Función de tabla MQRECEIVEALLCLOB" en las *Rutinas administrativas de SQL*
- "Mandato db2mqlsn - MQ Listener" en la publicación *Consulta de mandatos*
- "enable_MQFunctions" en la publicación *Consulta de mandatos*
- "disable_MQFunctions" en la publicación *Consulta de mandatos*

Recuperación de mensajes con funciones de WebSphere MQ

Utilizando las funciones de WebSphere® MQ DB2®, se pueden recibir o leer mensajes. La diferencia entre leer y recibir es que la operación de lectura devuelve el mensaje en la cabecera de una cola sin eliminarlo de la misma, mientras que las operaciones de recepción hacen que el mensaje se elimine de la cola. Cuando se utiliza una operación de recepción para recuperar un mensaje, el mismo mensaje sólo se puede recuperar una vez. Cuando se utiliza una operación de lectura para recuperar un mensaje, el mismo mensaje se puede recuperar varias veces. Los ejemplos siguientes muestran las operaciones de recuperación.

Ejemplo 9:

```
VALUES DB2MQ.MQREAD()
```

El "Ejemplo 9" en la página 553 devuelve una serie VARCHAR que contiene el mensaje en la cabecera de la cola definida por el servicio por omisión utilizando la política de calidad de servicio por omisión. Si no hay ningún mensaje disponible para su lectura, se devuelve un valor nulo. Esta operación no modifica la cola.

Ejemplo 10:

```
VALUES DB2MQ.MQRECEIVE('Employee_Changes')
```

El "Ejemplo 10" en la página 553 muestra cómo se puede eliminar un mensaje de la cabecera de la cola definida por el servicio Employee_Changes utilizando la política por omisión.

Una característica de DB2 es la posibilidad de generar una tabla a partir de una función definida por el usuario (o proporcionada por DB2). Puede aprovechar esta característica de función de tabla de modo que el contenido de una cola se materialice como una tabla de DB2 Universal Database™. El siguiente ejemplo muestra la forma más sencilla de esta característica:

Ejemplo 11:
```
SELECT t.* FROM table ( DB2MQ.MQREADALL()) t
```

La consulta del "Ejemplo 11" devuelve una tabla que consta de todos los mensajes de la cola definidos por el servicio por omisión y los metadatos sobre estos mensajes.

Para devolver únicamente los mensajes, se podría reescribir el ejemplo del siguiente modo:

Ejemplo 12:
```
SELECT t.MSG FROM table (DB2MQ.MQREADALL()) t
```

La tabla que devuelve una función de tabla no es distinta de una tabla recuperada directamente de una base de datos. Esto significa que puede utilizar esta tabla de varias formas. Por ejemplo, puede unir el contenido de la tabla con otra tabla o contar el número de mensajes de una cola:

Ejemplo 13:
```
SELECT t.MSG, e.LASTNAME FROM table (DB2MQ.MQREADALL() ) t, EMPLOYEE e
  WHERE t.MSG = e.LASTNAME
```

Ejemplo 14:
```
SELECT COUNT(*) FROM table (DB2MQ.MQREADALL()) t
```

también puede ocultar el hecho de que la fuente de la tabla es una cola creando una vista sobre una función de tabla. Por ejemplo, el ejemplo siguiente crea una vista denominada NEW_EMP sobre una cola a la que hace referencia el servicio denominado NEW_EMPLOYEES:

Ejemplo 15:
```
CREATE VIEW NEW_EMP (msg) AS SELECT t.msg
   FROM table(DB2MQ.MQREADALL(NEW_EMPLOYEES)) t
```

En este caso ("Ejemplo 15"), la vista está definida con una sola columna que contiene un mensaje entero. Si los mensajes están estructurados de forma sencilla, por ejemplo si contienen dos campos de longitud fija, resulta sencillo utilizar las funciones integradas de DB2 para analizar el mensaje en las dos columnas. Por ejemplo, si sabe que los mensajes enviados a una determinada cola siempre contienen un apellido de 18 caracteres seguido de un nombre de 18 caracteres, puede definir una vista que contenga cada campo como una columna separada del siguiente modo:

Ejemplo 16:
```
CREATE VIEW NEW_EMP2 AS SELECT left(t.msg,18) AS LNAME, right(t.msg,18) AS FNAME
   FROM table(DB2MQ.MQREADALL()) t
```

Se puede utilizar una función del Centro de desarrollo de DB2, el Asistente de ayuda de WebSphere MQ, para crear nuevas funciones y vistas de tablas de DB2 que correlacionan estructuras de mensajes delimitados con columnas.

Suele resultar práctico almacenar el contenido de uno o más mensajes en la base de datos. Esto se puede hacer utilizando toda la potencia de SQL para manipular y almacenar contenido de mensajes. Un ejemplo sencillo de esto se muestra a continuación:

Ejemplo 17:

```
INSERT INTO MESSAGES SELECT t.msg FROM table (DB2MQ.MQRECEIVEALL()) t
```

Dada una tabla MESSAGES, con una sola columna de VARCHAR(2000), la sentencia anterior inserta los mensajes procedentes de la cola de servicio por omisión en la tabla. Esta técnica se puede utilizar para cubrir una amplia gama de circunstancias.

Conceptos relacionados:

- "Habilitación de MQSeries" en la página 17
- "Visión general funcional de WebSphere MQ" en la página 547
- "Mensajería de WebSphere MQ" en la página 549
- "Envío de mensajes con funciones de WebSphere MQ" en la página 552
- "Conectividad de aplicación a aplicación de WebSphere MQ" en la página 555
- "Comunicaciones de solicitud/respuesta con funciones de WebSphere MQ" en la página 557
- "Publicación/suscripción con funciones de WebSphere MQ" en la página 558
- "Cómo utilizar funciones de WebSphere MQ en DB2" en la publicación *IBM DB2 Information Integrator Application Developer's Guide*

Tareas relacionadas:

- "Configuración de funciones de DB2 WebSphere MQ" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Información relacionada:

- "Función escalar MQSEND" en las *Rutinas administrativas de SQL*
- "Función escalar MQRECEIVE" en las *Rutinas administrativas de SQL*
- "Función escalar MQREAD" en las *Rutinas administrativas de SQL*
- "Función escalar MQPUBLISH" en las *Rutinas administrativas de SQL*
- "Función escalar MQSUBSCRIBE" en las *Rutinas administrativas de SQL*
- "Función escalar MQUNSUBSCRIBE" en las *Rutinas administrativas de SQL*
- "Función de tabla MQREADALL" en las *Rutinas administrativas de SQL*
- "Función de tabla MQRECEIVEALL" en las *Rutinas administrativas de SQL*
- "Función escalar MQREADCLOB" en las *Rutinas administrativas de SQL*
- "Función escalar MQRECEIVECLOB" en las *Rutinas administrativas de SQL*
- "Función de tabla MQREADALLCLOB" en las *Rutinas administrativas de SQL*
- "Función de tabla MQRECEIVEALLCLOB" en las *Rutinas administrativas de SQL*
- "Mandato db2mqlsn - MQ Listener" en la publicación *Consulta de mandatos*
- "enable_MQFunctions" en la publicación *Consulta de mandatos*
- "disable_MQFunctions" en la publicación *Consulta de mandatos*

Conectividad de aplicación a aplicación de WebSphere MQ

La integración de aplicaciones es un elemento común en muchas soluciones. Tanto si se integra una aplicación comprada en una infraestructura existente como si simplemente se integra una aplicación recién desarrollada en un entorno existente, generalmente nos enfrentamos a la tarea de unir un grupo heterogéneo de subsistemas para formar un entorno en funcionamiento.

WebSphere® MQ se suele considerar una herramienta esencial para integrar aplicaciones. Accesible en la mayoría de los entornos de hardware, software y lenguaje, WebSphere MQ proporciona los medios para interconectar un grupo muy heterogéneo de aplicaciones.

Consulte los temas ″Comunicaciones de solicitud/respuesta con funciones de WebSphere MQ″ y ″Publicación/suscripción con funciones de WebSphere MQ″ si desea ver casos de integración de aplicaciones y cómo se pueden utilizar con DB2®.

Conceptos relacionados:

- “Habilitación de MQSeries” en la página 17
- “Visión general funcional de WebSphere MQ” en la página 547
- “Mensajería de WebSphere MQ” en la página 549
- “Envío de mensajes con funciones de WebSphere MQ” en la página 552
- “Recuperación de mensajes con funciones de WebSphere MQ” en la página 553
- “Comunicaciones de solicitud/respuesta con funciones de WebSphere MQ” en la página 557
- “Publicación/suscripción con funciones de WebSphere MQ” en la página 558
- “Cómo utilizar funciones de WebSphere MQ en DB2” en la publicación *IBM DB2 Information Integrator Application Developer's Guide*

Tareas relacionadas:

- “Configuración de funciones de DB2 WebSphere MQ” en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Información relacionada:

- “Función escalar MQSEND” en las *Rutinas administrativas de SQL*
- “Función escalar MQRECEIVE” en las *Rutinas administrativas de SQL*
- “Función escalar MQREAD” en las *Rutinas administrativas de SQL*
- “Función escalar MQPUBLISH” en las *Rutinas administrativas de SQL*
- “Función escalar MQSUBSCRIBE” en las *Rutinas administrativas de SQL*
- “Función escalar MQUNSUBSCRIBE” en las *Rutinas administrativas de SQL*
- “Función de tabla MQREADALL” en las *Rutinas administrativas de SQL*
- “Función de tabla MQRECEIVEALL” en las *Rutinas administrativas de SQL*
- “Función escalar MQREADCLOB” en las *Rutinas administrativas de SQL*
- “Función escalar MQRECEIVECLOB” en las *Rutinas administrativas de SQL*
- “Función de tabla MQREADALLCLOB” en las *Rutinas administrativas de SQL*
- “Función de tabla MQRECEIVEALLCLOB” en las *Rutinas administrativas de SQL*
- “Mandato db2mqlsn - MQ Listener” en la publicación *Consulta de mandatos*
- “enable_MQFunctions” en la publicación *Consulta de mandatos*
- “disable_MQFunctions” en la publicación *Consulta de mandatos*

Comunicaciones de solicitud/respuesta con funciones de WebSphere MQ

El método de comunicaciones de solicitud/respuesta (R/R) es una técnica muy común para que una aplicación solicite los servicios de otra. Una forma de hacerlo es para que el solicitante envíe un mensaje al proveedor de servicio solicitando que se realice un trabajo. Una vez finalizado el trabajo, el proveedor puede decidir enviar los resultados (o simplemente una confirmación de finalización) al solicitante. Sin embargo, mediante las técnicas básicas de mensajería descritas anteriormente, no hay nada que conecte la solicitud del remitente con la respuesta del proveedor de servicio. A no ser que el solicitante espere una respuesta antes de continuar, se debe utilizar algún mecanismo para asociar cada respuesta con su solicitud. En lugar de obligar al desarrollador a crear dicho mecanismo, WebSphere® MQ proporciona un identificador de correlación que permite la correlación de mensajes en un intercambio.

Aunque hay varias formas de utilizar este mecanismo, la manera más sencilla consiste en que el solicitante marque un mensaje con un identificador de correlación conocido utilizando, por ejemplo, lo siguiente:

Ejemplo 18:

```
VALUES DB2MQ.MQSEND ('myRequester','myPolicy','SendStatus:cust1','Req1')
```

Esta sentencia añade un parámetro final Req1 a la sentencia MQSEND de arriba para indicar el identificador de correlación correspondiente a la solicitud. Para recibir una respuesta a esta solicitud específica, utilice la sentencia MQRECEIVE correspondiente para recuperar de forma selectiva el primer mensaje definido por el servicio indicado que coincide con el identificador de correlación del siguiente modo:

Ejemplo 19:

```
VALUES DB2MQ.MQRECEIVE('myReceiver','myPolicy','Req1')
```

Si la aplicación que da servicio a la solicitud está ocupada y el solicitante emite el MQRECEIVE anterior antes de que se envíe la respuesta, no se encuentra ningún mensaje que coincida con este identificador de correlación.

Para recibir tanto la solicitud de servicio como el identificador de correlación, utilice una sentencia como la siguiente:

Ejemplo 20:

```
SELECT msg, correlid
  FROM table (VALUES DB2MQ.MQRECEIVEALL('aServiceProvider','myPolicy',1)) t
```

Esto devuelve el mensaje y el identificador de correlación de la primera solicitud procedente del servicio aServiceProvider.

Una vez realizado el servicio, este envía el mensaje de respuesta a la cola descrita por aRequester. Mientras tanto, el solicitante del servicio puede estar haciendo otro trabajo. De hecho, no hay ninguna garantía de que la solicitud de servicio inicial reciba una respuesta en un límite de tiempo definido. El desarrollador debe gestionar los tiempos de espera excedidos a nivel de aplicación como este; el solicitante debe realizar un sondeo para detectar la presencia de la respuesta. La ventaja de este proceso asíncrono que no depende del tiempo es que el solicitante y el proveedor de servicio se ejecutan de forma completamente independiente uno del otro. Este método se puede utilizar en entornos en los que las aplicaciones sólo se conectan de forma intermitente y en entornos por lotes en los que se agregan varias solicitudes o respuestas antes de su proceso. Este tipo de agregación se suele

utilizar en entornos de depósito de datos para actualizar de forma periódica un depósito de datos o un almacén de datos operativos.

Conceptos relacionados:

- "Habilitación de MQSeries" en la página 17
- "Visión general funcional de WebSphere MQ" en la página 547
- "Mensajería de WebSphere MQ" en la página 549
- "Envío de mensajes con funciones de WebSphere MQ" en la página 552
- "Recuperación de mensajes con funciones de WebSphere MQ" en la página 553
- "Conectividad de aplicación a aplicación de WebSphere MQ" en la página 555
- "Publicación/suscripción con funciones de WebSphere MQ" en la página 558
- "Cómo utilizar funciones de WebSphere MQ en DB2" en la publicación *IBM DB2 Information Integrator Application Developer's Guide*

Tareas relacionadas:

- "Configuración de funciones de DB2 WebSphere MQ" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Información relacionada:

- "Función escalar MQSEND" en las *Rutinas administrativas de SQL*
- "Función escalar MQRECEIVE" en las *Rutinas administrativas de SQL*
- "Función escalar MQREAD" en las *Rutinas administrativas de SQL*
- "Función escalar MQPUBLISH" en las *Rutinas administrativas de SQL*
- "Función escalar MQSUBSCRIBE" en las *Rutinas administrativas de SQL*
- "Función escalar MQUNSUBSCRIBE" en las *Rutinas administrativas de SQL*
- "Función de tabla MQREADALL" en las *Rutinas administrativas de SQL*
- "Función de tabla MQRECEIVEALL" en las *Rutinas administrativas de SQL*
- "Función escalar MQREADCLOB" en las *Rutinas administrativas de SQL*
- "Función escalar MQRECEIVECLOB" en las *Rutinas administrativas de SQL*
- "Función de tabla MQREADALLCLOB" en las *Rutinas administrativas de SQL*
- "Función de tabla MQRECEIVEALLCLOB" en las *Rutinas administrativas de SQL*
- "Mandato db2mqlsn - MQ Listener" en la publicación *Consulta de mandatos*
- "enable_MQFunctions" en la publicación *Consulta de mandatos*
- "disable_MQFunctions" en la publicación *Consulta de mandatos*

Publicación/suscripción con funciones de WebSphere MQ

dos técnicas de la mensajería de datos son la publicación simple de datos y la publicación automática.

Publicación simple de datos

Un escenario común en la integración de aplicaciones es aquella en la que una aplicación notifica a otras aplicaciones sobre sucesos de interés. Esto se puede hacer fácilmente enviando un mensaje a una cola supervisada por otra aplicación. El contenido del mensaje puede ser una serie definida por el usuario o puede ser una composición a partir de columnas de base de datos. A menudo, un menaje sencillo es todo lo que se tiene que enviar mediante la función MQSEND. Cuando se tiene que enviar este tipo de mensaje simultáneamente a varios destinatarios, se puede utilizar la función de Lista de distribución de WebSphere® MQ AMI.

Una lista de definición se define mediante la herramienta de administración de AMI. Una lista de distribución contiene una lista de servicios individuales. Un mensaje enviado a una lista de distribución se reenvía a cada servicio definido dentro de la lista. Esto resulta especialmente útil cuando se sabe que unos cuantos servicios siempre están interesados en todos los mensajes. El ejemplo siguiente muestra el envío de un mensaje a la lista de distribución interestedParties:

Ejemplo 21:

```
VALUES DB2MQ.MQSEND('interestedParties', 'information of general interest');
```

Cuando se necesita un mayor control sobre los mensajes que deben recibir determinados servicios, se necesita la función de Publicación/suscripción. Los sistemas de Publicación/suscripción suelen proporcionar un entorno seguro y escalable en el que muchos suscriptores se pueden registrar para recibir mensajes de muchos publicadores. Para dar soporte a esta función, se puede utilizar la interfaz MQPublish junto con MQSeries® Integrator o con el recurso de Publicación/suscripción de WebSphere MQ.

MQPublish permite a los usuarios especificar opcionalmente un tema que se asociará a un mensaje. Los temas permiten al suscriptor especificar más claramente los mensajes que deben aceptar. La secuencia de pasos a seguir es la siguiente:

1. Un administrador de WebSphere MQ configura las funciones de publicación/suscripción de MQSeries Integrator.
2. Las aplicaciones interesadas se suscriben a los puntos de suscripción definidos por la configuración MQSI y pueden especificar si lo desean temas que les interesan. Cada suscriptor selecciona temas relevantes y también puede utilizar las técnicas de suscripción basadas en contenido de MQSeries Integrator Versión 2. Es importante tener en cuenta que las colas, representadas por nombres de servicio, definen al suscriptor.
3. Una aplicación DB2® publica un mensaje en el punto de servicio Weather. El mensaje indica que el clima (weather) es ″Sleet″ y su tema es Austin, lo que notifica a los suscriptores interesados que el clima en Austin es Sleet.
4. La mecánica para publicar realmente el mensaje la manejan las funciones de WebSphere MQ que proporciona DB2. El mensaje se envía a MQSeries con el servicio denominado Weather.
5. MQSI acepta el mensaje procedente del servicio Weather, realiza el proceso definido por la configuración de MQSI y determina qué suscripciones satisface. Luego MQSI reenvía el mensaje a las colas de suscriptores cuyos criterios cumple.
6. Las aplicaciones que se han suscrito para el servicio Weather y que han registrado su interés en Austin recibirán el mensaje Sleet en su servicio de recepción.

Para publicar estos datos utilizando los valores por omisión y un tema nulo, utilice la siguiente sentencia:

Ejemplo 22:

```
SELECT DB2MQ.MQPUBLISH (LASTNAME || ' ' || FIRSTNAME || ' ' || DEPARTMENT
  ||' ' || char(AGE)) FROM EMPLOYEE WHERE DEPARTMENT = '5LGA'
```

Si se desea especificar por completo todos los parámetros y simplificar el mensaje para que contenga únicamente LASTNAME, la sentencia es la siguiente:

Ejemplo 23:

```
SELECT DB2MQ.MQPUBLISH('HR_INFO_PUB', 'SPECIAL_POLICY', LASTNAME,
  'ALL_EMP:5LGA', 'MANAGER') FROM EMPLOYEE WHERE DEPARTMENT = '5LGA'
```

Esta sentencia publica mensajes en el servicio de publicación HR_INFO_PUB utilizando el servicio SPECIAL_POLICY. Los mensajes indican que el remitente es el identificador de correlación MANAGER. La serie del tema ("ALL_EMP:5LGA") muestra que puede especificar varios temas, concatenándolos con un signo de dos puntos (":"). En este ejemplo, el uso de dos temas permite a los suscriptores registrarse para ALL_EMP o sólo para 5LGA para recibir estos mensajes. Para recibir mensajes publicados, primero debe registrar su interés en mensajes que contengan un determinado tema e indicar el nombre del servicio de suscriptor al que se deben enviar los mensajes. Es importante tener en cuenta que un servicio de suscriptor AMI define un servicio intermediario y un servicio de receptor. El servicio intermediario constituye el modo en que el suscriptor se comunica con el intermediario de publicación/suscripción y el servicio de receptor especifica dónde se enviarán los mensajes que coincidan con la solicitud de suscripción. La siguiente sentencia registra un interés en el tema ALL_EMP.

Ejemplo 24:
```
VALUES DB2MQ.MQSUBSCRIBE('aSubscriber', 'ALL_EMP')
```

Una vez se ha suscrito una aplicación, los mensajes publicados con el tema ALL_EMP se reenvían al servicio de receptor definido por el servicio de suscriptor. Una aplicación puede tener varias suscripciones simultáneas. Para obtener los mensajes que cumplen con la suscripción, puede utilizar cualquiera de las funciones estándares de recuperación de mensajes. Por ejemplo, si el servicio de suscriptor aSubscriber define que el servicio de receptor es aSubscriberReceiver, la sentencia lee el primer mensaje de forma no destructiva:

Ejemplo 25:
```
VALUES DB2MQ.MQREAD('aSubscriberReceiver')
```

Para determinar tanto los mensajes como los temas bajo los que se han publicado, utilice una de las funciones de tabla. La sentencia siguiente recibe los cinco primeros mensajes procedentes de aSubscriberReceiver y muestra tanto el mensaje como el tema:

Ejemplo 26:
```
SELECT t.msg, t.topic FROM table (DB2MQ.MQRECEIVEALL('aSubscriberReceiver',5)) t
```

Para leer todos los mensajes con el tema ALL_EMP, puede aprovechar la potencia de SQL emitiendo la siguiente sentencia:

Ejemplo 27:
```
SELECT t.msg FROM table (DB2MQ.MQREADALL('aSubscriberReceiver')) t
  WHERE t.topic = 'ALL_EMP'
```

Nota: Es importante observar que si se utiliza MQRECEIVEALL con una restricción, se consumirá la cola completa, no únicamente los mensajes publicados con el tema ALL_EMP. Esto se debe a que la función de tabla se realiza antes de que se aplique la restricción.

Cuando ya no esté interesado en suscribirse a un determinado tema, debe eliminar la suscripción de forma explícita con una sentencia como la siguiente:

Ejemplo 28:
```
VALUES DB2MQ.MQUNSUBSCRIBE('aSubscriber', 'ALL_EMP')
```

Después de emitir la sentencia anterior, el intermediario de publicación/suscripción deja de distribuir mensajes que coinciden con esta suscripción.

Publicación automática

Otra técnica importante de la mensajería de bases de datos es la publicación automática. Mediante el recurso de activación de DB2, puede publicar automáticamente mensajes como parte de una invocación de activación. Aunque existen otras técnicas para la publicación automática de datos, el enfoque basado en activador ofrece a los administradores o desarrolladores una gran libertad para construir el contenido del mensaje y flexibilidad para definir las acciones de activación. Al igual que sucede con cualquier uso que se haga de activadores, debe saber la frecuencia y el coste de la ejecución. Los ejemplos siguientes muestran cómo se pueden utilizar activadores con las funciones de WebSphere MQ DB2.

El ejemplo siguiente muestra lo fácil que resulta publicar un mensaje cada vez que se contrata un nuevo empleado. Cualquier usuario o aplicación suscrito al servicio HR_INFO_PUB con interés registrado en NEW_EMP recibirá un mensaje que contendrá la fecha, nombre y departamento de cada nuevo empleado.

Ejemplo 29:

```
CREATE TRIGGER new_employee AFTER INSERT ON employee
  REFERENCING NEW AS n FOR EACH ROW MODE DB2SQL
  VALUES DB2MQ.MQPUBLISH('HR_INFO_PUB',current date||''||
    'LASTNAME'||''DEPARTMENT,'NEW_EMP')
```

Conceptos relacionados:

- "Habilitación de MQSeries" en la página 17
- "Visión general funcional de WebSphere MQ" en la página 547
- "Mensajería de WebSphere MQ" en la página 549
- "Envío de mensajes con funciones de WebSphere MQ" en la página 552
- "Recuperación de mensajes con funciones de WebSphere MQ" en la página 553
- "Conectividad de aplicación a aplicación de WebSphere MQ" en la página 555
- "Comunicaciones de solicitud/respuesta con funciones de WebSphere MQ" en la página 557
- "Cómo utilizar funciones de WebSphere MQ en DB2" en la publicación *IBM DB2 Information Integrator Application Developer's Guide*

Tareas relacionadas:

- "Configuración de funciones de DB2 WebSphere MQ" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Información relacionada:

- "Función escalar MQSEND" en las *Rutinas administrativas de SQL*
- "Función escalar MQRECEIVE" en las *Rutinas administrativas de SQL*
- "Función escalar MQREAD" en las *Rutinas administrativas de SQL*
- "Función escalar MQPUBLISH" en las *Rutinas administrativas de SQL*
- "Función escalar MQSUBSCRIBE" en las *Rutinas administrativas de SQL*
- "Función escalar MQUNSUBSCRIBE" en las *Rutinas administrativas de SQL*
- "Función de tabla MQREADALL" en las *Rutinas administrativas de SQL*
- "Función de tabla MQRECEIVEALL" en las *Rutinas administrativas de SQL*
- "Función escalar MQREADCLOB" en las *Rutinas administrativas de SQL*
- "Función escalar MQRECEIVECLOB" en las *Rutinas administrativas de SQL*
- "Función de tabla MQREADALLCLOB" en las *Rutinas administrativas de SQL*
- "Función de tabla MQRECEIVEALLCLOB" en las *Rutinas administrativas de SQL*

- "Mandato db2mqlsn - MQ Listener" en la publicación *Consulta de mandatos*
- "enable_MQFunctions" en la publicación *Consulta de mandatos*
- "disable_MQFunctions" en la publicación *Consulta de mandatos*

Capítulo 25. WebSphere

Las secciones siguientes describen la agrupación de conexiones de WebSphere y la puesta de sentencias en antememoria.

Conexión con datos de la empresa

Muchas empresas guardan sus datos en grandes sistemas, tales como servidores zSeries®.Las aplicaciones Web necesitan medios para acceder a esos datos.

DB2® Connect proporciona a las aplicaciones basadas en Windows® o UNIX® la capacidad de conectar con datos almacenados en esos grandes sistemas y de poderlos utilizar. DB2 también ofrece su propio conjunto de funciones como la agrupación de conexiones y el concentrador de conexiones.

Fuentes de datos y agrupación de conexiones WebSphere

Cada vez que un recurso intenta acceder a una base de datos, debe establecer conexión con dicha base de datos. Una conexión con una base de datos implica actividad general; necesita recursos para crear la conexión, mantenerla y luego liberarla cuando ya no la necesita.

Nota: La información que se proporciona aquí es aplicable a la Versión 4 de WebSphere® Application Server para UNIX®, LINUX y Windows®.

La actividad general total de la base de datos correspondiente a una aplicación es especialmente alta para aplicaciones basadas en la Web, porque los usuarios de la Web se conectan y desconectan con más frecuencia. Además, las interacciones de los usuarios suelen ser más cortas, debido a la naturaleza de Internet. Normalmente, se emplea más esfuerzo en conectar y desconectar que durante la propia transacción. Además, puesto que las peticiones de Internet pueden llegar prácticamente de cualquier lugar, los volúmenes de uso pueden ser grandes y difíciles de predecir.

IBM® WebSphere Application Server permite a los administradores crear una agrupación de conexiones de base de datos que pueden ser compartidas por aplicaciones de un servidor de aplicaciones para solucionar estos problemas de actividad general.

La agrupación de conexiones extiende la actividad general de conexión entre varias peticiones de usuarios, por lo que conserva los recursos para futuras peticiones.

Puede utilizar la agrupación de conexiones de WebSphere o el soporte de agrupación de conexiones de DB2®, que se proporciona mediante la API Paquete opcional de JDBC 2.1, para establecer la agrupación de conexiones.

La agrupación de conexiones de WebSphere es la implantación de la especificación de la API Paquete opcional de JDBC 2.1. Por lo tanto, el modelo de programación de agrupación de conexiones es el que se especifica en las especificaciones JDBC 2.1 Core y API Paquete opcional de JDBC 2.1. Esto significa que las aplicaciones que obtienen sus conexiones a través de una fuente de datos creada en WebSphere

Application Server pueden aprovechar las funciones de JDBC 2.1 como la agrupación de conexiones y las conexiones habilitadas para JTA.

Además, la agrupación de conexiones de WebSphere proporciona funciones adicionales que permiten a los administradores ajustar la agrupación para optimizar su rendimiento y proporcionar aplicaciones con excepciones de WebSphere que permiten a los programadores escribir aplicaciones sin conocer las SQLExceptions comunes específicas del proveedor. No todas las SQLExceptions específicas del proveedor se correlacionan con excepciones de WebSphere; se deben codificar las aplicaciones para que puedan manejar las SQLExceptions específicas del proveedor. Sin embargo, las excepciones recuperables más comunes se correlacionan con excepciones de WebSphere.

La fuente de datos obtenida a través de WebSphere es una fuente de datos que implanta la API Paquete opcional de JDBC 2.1. Proporciona agrupación de conexiones y, en función de la fuente de datos seleccionada específica del proveedor, es posible que proporcione conexiones capaces de participar en transacciones de protocolo de confirmación en dos fases (habilitadas para JTA).

El programa `AccessEmployee` del archivo `AccessEmployee.ear` utiliza `DataSource` de WebSphere para acceder a una base de datos DB2.

Información relacionada:

- "Ejemplos de Java WebSphere" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Ventajas de la agrupación de conexiones de WebSphere

La agrupación de conexiones puede mejorar el tiempo de respuesta de cualquier aplicación que necesite conexiones, especialmente aplicaciones basadas en la Web.

Cuando un usuario realiza una petición a un recurso sobre la Web, el recurso accede a una fuente de datos. La mayoría de las peticiones de usuario no implican la actividad general de crear una nueva conexión, porque la fuente de datos puede localizar y utilizar una conexión existente de la agrupación de conexiones. Cuando la petición se responde y la respuesta se devuelve al usuario, el recurso devuelve la conexión a la agrupación de conexiones para que se reutilice. De nuevo, se evita la actividad general de una desconexión.

Cada usuario asimila una facción del coste de conexión o desconexión. Una vez se han utilizado los recursos iniciales para generar las conexiones de la agrupación, la actividad general adicional es insignificante porque se reutilizan las conexiones existentes.

La colocación en antememoria de sentencias preparadas es otro mecanismo por el cual el método de agrupación de conexiones de WebSphere® mejora los tiempos de respuesta de aplicaciones basadas en la Web.

Una antememoria de sentencias preparadas previamente está disponible en una conexión. Cuando se solicita una nueva sentencia preparada en una conexión, se devuelve la sentencia preparada colocada en antememoria, si está disponible. Esta colocación en antememoria reduce el número de sentencias preparadas creadas, que resultan caras, lo cual mejora los tiempos de respuesta. La antememoria resulta útil para aplicaciones que tienden a preparar la misma sentencia varias veces.

Además de mejorar los tiempos de respuesta, el método de agrupación de conexiones de WebSphere proporciona una capa de abstracción de la base de datos, que puede colocar en almacenamiento intermedio la aplicación cliente y hacer que parezca que distintas bases de datos funcionan del mismo modo ante una aplicación.

Esta colocación en almacenamiento intermedio facilita la conmutación de bases de datos de aplicación, puesto que el código de la aplicación no tiene que manejar SQLExceptions comunes específicas del cliente, sino una excepción de agrupación de conexiones de WebSphere.

Colocación de sentencias en antememoria en WebSphere

WebSphere® proporciona un mecanismo para colocar en antememoria sentencias preparadas previamente. La colocación en antememoria de sentencias preparadas mejora los tiempos de respuesta, ya que una aplicación puede reutilizar una `PreparedStatement` en una conexión si existe en la antememoria de dicha conexión, sin tener que crear una nueva `PreparedStatement`.

Cuando una aplicación crea una `PreparedStatement` en una conexión, primero se busca en la antememoria de la conexión para determinar si ya existe una `PreparedStatement` con la misma serie SQL. Esta búsqueda se realiza utilizando la serie entera de sentencias de SQL en el método `prepareStatement()`. Si se encuentra una coincidencia, se devuelve la PreparedStatement colocada en antememoria para que se pueda utilizar. Si no se encuentra, se crea una nueva `PreparedStatement` y se devuelve a la aplicación.

A medida que la aplicación cierra sentencias preparadas, estas se devuelven a la antememoria de sentencias de la conexión. Por omisión, sólo se pueden mantener 100 sentencias preparadas en antememoria para la agrupación entera de conexiones. Por ejemplo, si hay diez conexiones en la agrupación, el número de sentencias preparadas colocadas en antememoria correspondientes a estas diez conexiones no puede ser mayor que 100. Esto asegura que hay un número limitado de sentencias preparadas simultáneamente abiertas para la base de datos, lo que ayuda a evitar problemas de recursos con una base de datos.

Los elementos sólo se eliminan de la antememoria de sentencias preparadas de la conexión cuando el número de sentencias preparadas colocadas simultáneamente en antememoria supera el valor de `statementCacheSize` (que por omisión es 100). Si una sentencia preparada se tiene que eliminar de la antememoria, se elimina y se añade a un vector de sentencias descartadas. En cuanto finaliza el método en el que se ha eliminado la sentencia preparada, las sentencias preparadas del vector de sentencias descartadas se cierran para la base de datos. Por lo tanto, en un determinado momento, puede haber 100 más el número de sentencias recientemente descartadas abiertas para la base de datos. Las sentencias preparadas adicionales se cierran cuando el método finaliza.

El número de sentencias preparadas que se pueden colocar en antememoria se puede configurar en la fuente de datos. Cada antememoria se debe ajustar según los requisitos de la aplicación en cuanto a sentencias preparadas.

Parte 6. Plug-ins de seguridad

Capítulo 26. Plug-ins de seguridad

Plug-ins de seguridad 569
Ubicaciones de las bibliotecas de plug-in de seguridad 573
Convenios para nombrar plug-ins de seguridad 574
Soporte de plug-in de seguridad para los ID de usuario de dos componentes 575
Consideraciones sobre los sistemas de 32 y 64 bits para los plug-ins de seguridad 578
Determinación de problemas para plug-ins de seguridad 578
Despliegue de un plug-in de recuperación de grupos 580
Despliegue de un plug-in de ID de usuario/contraseña 581
Despliegue de un plug-in de GSS-API 583
Despliegue de un plug-in Kerberos 584

Plug-ins de seguridad

En DB2®, la autenticación se realiza utilizando *plug-ins de seguridad*. Un plug-in de seguridad es un biblioteca de carga dinámica que DB2 carga para proporcionar las funciones siguientes:

- plug-in de recuperación de grupos: recupera información sobre pertenencia a grupos para un usuario determinado
- plug-in de autenticación de cliente: gestiona la autenticación en un cliente DB2
- plug-in de autenticación de servidor: gestiona la autenticación en un servidor DB2

DB2 soporta dos mecanismos para la autenticación por plug-in:

- Autenticación mediante un ID de usuario y contraseña, que se denomina autenticación por ID de usuario/contraseña. Los tipos de autenticación CLIENT, SERVER, SERVER_ENCRYPT, DATA_ENCRYPT y DATA_ENCRYPT_CMP determinan cómo y cuándo se produce la autenticación de un usuario. El tipo de autenticación utilizado es el especificado en el parámetro de configuración *authentication* de la base de datos. Todos estos tipos de autenticación se implementan utilizando plug-ins de autenticación por ID de usuario/contraseña.
- La autenticación mediante GSS-API, conocida formalmente como *Generic Security Service Application Program Interface, Versión 2* (IETF RFC2743) y *Generic Security Service API Versión 2: C-Bindings* (IETF RFC2744). La autenticación Kerberos también se implementa utilizando GSS-API. Los tipos de autenticación KERBEROS, GSSPLUGIN, KRB_SERVER_ENCRYPT y GSS_SERVER_ENCRYPT utilizan plug-ins de autenticación GSS-API. KRB_SERVER_ENCRYPT y GSS_SERVER_ENCRYPT soportan la autenticación GSS-API y la autenticación por ID de usuario/contraseña, pero la autenticación GSS-API es el tipo preferido.

Cada plug-in se puede utilizar por separado o en combinación con uno o más de los demás plug-ins. Por ejemplo, puede utilizar solo un plug-in de autenticación de servidor y aceptar los valores por omisión de DB2 para la autenticación de cliente y de grupo. Como alternativa, puede utilizar solamente un plug-in de autenticación de grupo o de cliente. El único caso en el que son necesarios tanto un plug-in de cliente como un plug-in de servidor es para los plug-ins de autenticación GSS-API.

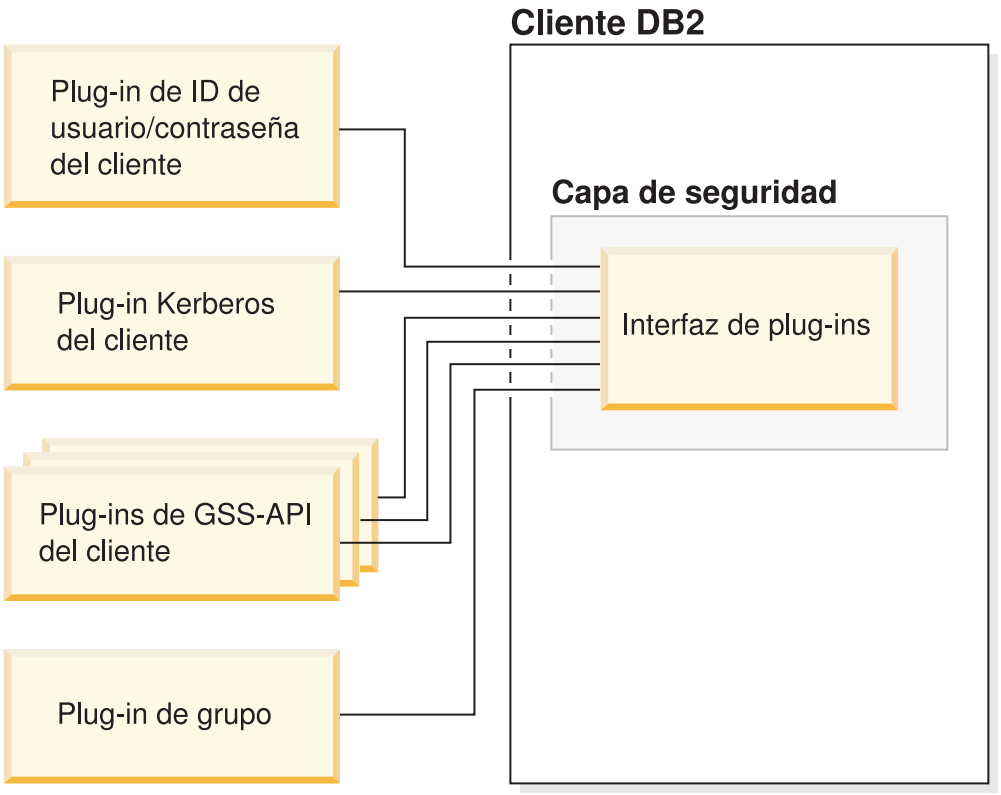
En DB2 Versión 8.2, la opción por omisión es utilizar un plug-in de ID de usuario/contraseña que implementa un mecanismo de autenticación a nivel del sistema operativo. En todos los releases anteriores de DB2 la opción por omisión es utilizar directamente la autenticación a nivel del sistema operativo sin implementar

un plug-in. En DB2 Versión 8.2 se proporciona soporte Kerberos para el cliente en los sistemas operativos Solaris, AIX®, Windows® e IA32 Linux, pero solo está habilitado por omisión en Windows.

DB2 incluye conjuntos de plug-ins para la recuperación de grupos, la autenticación por ID de usuario/contraseña y la autenticación Kerberos. La arquitectura del plug-in de seguridad le permite personalizar la función de autenticación de DB2 desarrollando sus propios plug-ins o adquiriendo plug-ins de terceros.

Despliegue de plug-ins de seguridad en clientes DB2:

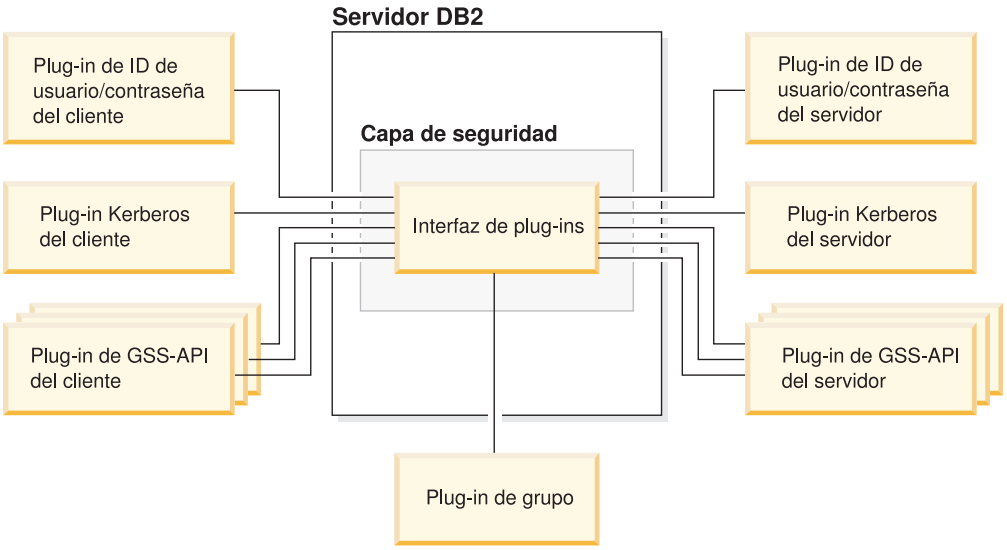
Los clientes DB2 pueden soportar un plug-in de grupo, un plug-in de autenticación por ID de usuario/contraseña, y negociar un plug-in de GSS-API determinado con el servidor DB2. En esta negociación, el cliente examina la lista de plug-ins de GSS-API implementados del servidor DB2 y busca el primer nombre de plug-in de autenticación que coincida con un plug-in de autenticación implementado en el cliente. La lista de plug-ins del servidor es un valor especificado por el usuario en un parámetro de configuración del gestor de bases de datos y contiene los nombres de todos los plug-ins que están implementados en el servidor. La figura siguiente muestra la infraestructura del plug-in de seguridad en un cliente DB2.



Despliegue de plug-ins de seguridad en servidores DB2:

Los servidores DB2 pueden soportar un plug-in de grupo, un plug-in de autenticación por ID de usuario/contraseña y varios plug-ins de GSS-API. Los diversos plug-ins de GSS-API se especifican en forma de lista como valor de un parámetro de configuración del gestor de bases de datos. Un solo plug-in de GSS-API de esa lista puede ser un plug-in Kerberos.

Además de desplegar plug-ins de seguridad del servidor, puede también desplegar plug-ins de autorización del cliente en su servidor de bases de datos. Cuando el usuario ejecuta operaciones a nivel de instancia, tales como db2start y db2trc, DB2 comprueba la autorización para la operación utilizando plug-ins de autenticación del cliente. Por tanto, debe instalar una versión de cliente del plug-in de autenticación cuyo tipo sea el especificado en el parámetro de configuración *authentication* del gestor de bases de datos. Si no despliega plug-ins de autenticación de cliente en el servidor de bases de datos, las operaciones a nivel de instancia, tales como db2start, no serán efectivas. Por ejemplo, si el tipo de autenticación es SERVER y no se utiliza ningún plug-in de cliente proporcionado por el usuario, DB2 recurre al plug-in de cliente por omisión proporcionado por IBM, que actúa a nivel del sistema operativo. La figura siguiente muestra la infraestructura del plug-in de seguridad en un servidor DB2.



Habilitación de los plug-ins de seguridad:

El administrador del sistema puede especificar los nombres de los plug-ins que se deben utilizar para cada mecanismo de autenticación; para ello actualiza determinados parámetros de configuración del gestor de bases de datos que están asociados a plug-ins. Si estos parámetros son nulos, se utilizan por omisión los plug-ins proporcionados por DB2 para la recuperación de grupos, la gestión por ID de usuario/contraseña, o Kerberos (si autenticación está definida como Kerberos (extremo servidor solamente)). En cambio, DB2 no proporciona un valor por omisión para el plug-in de GSS-API. Por tanto, si el administrador del sistema especifica el tipo de autenticación GSSPLUGIN en el parámetro "authentication", debe también especificar un plug-in de autenticación en *srvcon_gssplugin_list*.

Carga de plug-ins de seguridad por DB2:

En el servidor DB2, cuando se inicia el gestor de bases de datos, se cargan todos los plug-ins soportados que están especificados en los parámetros de configuración del gestor de bases de datos.

Para conexiones lógicas de base de datos y conexiones físicas de instancia, el cliente DB2 carga el plug-in apropiado de acuerdo con el mecanismo de seguridad negociado con el servidor durante las operaciones de conexión lógica o física. Una aplicación cliente puede también hacer que se carguen y utilicen simultáneamente

varios plug-ins de seguridad. Esto puede suceder, por ejemplo, en un programa multihebra que tiene conexiones simultáneas con diversas bases de datos desde instancias diferentes.

Para las acciones que no sean conexiones lógicas de base de datos ni conexiones físicas de instancia que requieran autorización (tales como actualizar la configuración de la base de datos, iniciar y detener el gestor de bases de datos, activar y desactivar la función de rastreo de DB2), el programa cliente DB2 carga un plug-in especificado en otro parámetro de configuración del gestor de bases de datos. Si *authentication* tiene el valor GSSPLUGIN, DB2 utiliza el plug-in especificado por *local_gssplugin*. Si *authentication* tiene el valor KERBEROS, DB2 utiliza el plug-in especificado por *clnt_krb_plugin*. En otro caso, DB2 utiliza el plug-in especificado por *clnt_pw_plugin*.

Desarrollo de plug-ins de seguridad:

Si está desarrollando un plug-in de seguridad, es necesario que implemente las funciones de autenticación estándar que serán invocadas por DB2. Para los tipos de plug-ins disponibles, la funcionalidad que necesita implementar es la siguiente:

Recuperación de grupos
Obtener la lista de grupos a los que pertenece un usuario.

Autenticación por ID de usuario/contraseña
Autenticación que identifica el contexto de seguridad por omisión (cliente solamente), valida y opcionalmente cambia una contraseña, determina si una cadena de caracteres proporcionada representa un usuario válido (servidor solamente), modifica el ID de usuario o contraseña proporcionados en el cliente antes de enviarlos al servidor (cliente solo) y devuelve el ID de autorización de DB2 asociado a un usuario determinado.

Autenticación GSS-API
Autenticación que implementa las funciones de GSS-API necesarias, identifica el contexto de seguridad por omisión (cliente solamente), genera las credenciales iniciales basándose en un ID de usuario y contraseña y opcionalmente cambia la contraseña (cliente solamente), crea y acepta certificados de seguridad, y devuelve el ID de autorización de DB2 asociado a un contexto de seguridad determinado de GSS-API.

Conceptos relacionados:

- "Authentication methods for your server" en la publicación *Administration Guide: Implementation*
- "Ubicaciones de las bibliotecas de plug-in de seguridad" en la página 573
- "Carga de plug-ins de seguridad por DB2" en la página 587
- "Las API de plug-in de seguridad" en la página 597

Información relacionada:

- "authentication - Authentication type configuration parameter" en la publicación *Administration Guide: Performance*
- "srvcon_auth - Authentication type for incoming connections at the server configuration parameter" en la publicación *Administration Guide: Performance*
- "Ejemplos del conector de seguridad" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Ubicaciones de las bibliotecas de plug-in de seguridad

Después obtener sus plug-ins de seguridad (ya sea desarrollando sus propios plug-ins o adquiriéndolos a terceros), debe copiarlos en lugares determinados del servidor de bases de datos.

DB2® busca plug-ins de autenticación del usuario para el extremo cliente en el directorio siguiente:

- UNIX® de 32 bits: `$DB2PATH/security32/plugin/client`
- UNIX de 64 bits: `$DB2PATH/security64/plugin/client`
- WINDOWS de 32 y 64 bits: `$DB2PATH\security\plugin\<nombre de instancia>\client`

Nota: En Windows®, los subdirectorios <nombre de instancia> y ″client″ no se crean automáticamente. El propietario de la instancia debe crearlos manualmente.

DB2 busca plug-ins de autenticación del usuario para el extremo servidor en el directorio siguiente:

- UNIX de 32 bits: `$DB2PATH/security32/plugin/server`
- UNIX de 64 bits: `$DB2PATH/security64/plugin/server`
- WINDOWS de 32 y 64 bits: `$DB2PATH\security\plugin\<nombre de instancia>\server`

Nota: En Windows, los subdirectorios <nombre de instancia> y ″server″ no se crean automáticamente. El propietario de la instancia debe crearlos manualmente.

DB2 busca plug-ins de grupo en el directorio siguiente:

- UNIX de 32 bits: `$DB2PATH/security32/plugin/group`
- UNIX de 64 bits: `$DB2PATH/security64/plugin/group`
- WINDOWS de 32 y 64 bits: `$DB2PATH\security\plugin\<nombre de instancia>\group`

Nota: En Windows, los subdirectorios <nombre de instancia> y ″group″ no se crean automáticamente. El propietario de la instancia debe crearlos manualmente.

Conceptos relacionados:

- “Plug-ins de seguridad” en la página 569
- “Carga de plug-ins de seguridad por DB2” en la página 587

Tareas relacionadas:

- “Despliegue de un plug-in de recuperación de grupos” en la página 580
- “Despliegue de un plug-in de ID de usuario/contraseña” en la página 581
- “Despliegue de un plug-in de GSS-API” en la página 583
- “Despliegue de un plug-in Kerberos” en la página 584

Información relacionada:

- “Restricciones para bibliotecas de plug-in de seguridad” en la página 588

Convenios para nombrar plug-ins de seguridad

Las bibliotecas de plug-in de seguridad deben tener la extensión de nombre de archivo apropiada para cada plataforma. Estas son las extensiones de archivo para cada sistema operativo:

- Windows®: .DLL
- AIX®: .a
- Linux, HP IPF y Solaris: .so
- HPUX sobre PA-RISC: .sl

Por ejemplo, suponga que tiene una biblioteca de plug-in de seguridad llamada `MyPlugin`. Para cada sistema operativo soportado, el nombre de archivo apropiado para la biblioteca es el siguiente:

- Windows de 32 bits: MyPlugin.dll
- Windows de 64 bits: MyPlugin64.dll
- AIX de 32 o 64 bits: MyPlugin.a
- SUN de 32 o 64 bits, Linux de 32 o 64 bits, HP de 32 o 64 bits sobre IPF: MyPlugin.so
- HP-UX de 32 o 64 bits sobre PA-RISC: MyPlugin.sl

Nota: El sufijo ″64″ solo es necesario en el nombre de biblioteca para los plug-ins de seguridad de Windows de 64 bits.

Cuando especifique el nombre del plug-in de seguridad en la configuración del gestor de bases de datos, debe utilizar el nombre completo de la biblioteca sin el sufijo ″64″ y omitir la extensión de archivo y cualquier vía de acceso calificada incluida en el nombre. Para cualquier sistema operativo, la biblioteca de plug-in de seguridad llamada `MyPlugin` se registraría de esta manera:

```
UPDATE DBM CFG USING CLNT_PW_PLUGIN MyPlugin
```

El nombre del plug-in de seguridad distingue entre mayúsculas y minúsculas, y debe coincidir exactamente con el nombre de biblioteca. DB2® utiliza el valor especificado en el parámetro de configuración del gestor de bases de datos para formar la vía de acceso de la biblioteca, y luego utiliza esa vía de acceso para cargar la biblioteca de plug-in de seguridad.

Para evitar conflictos en los nombres de los plug-ins de seguridad, es recomendable que designe el plug-in basándose en el método de autenticación utilizado y un símbolo identificador del fabricante del plug-in. Por ejemplo, si la empresa `Foo, Inc.` escribió un plug-in que hace uso del método de autenticación `somemethod`, el nombre del plug-in podría tener este aspecto: `FOOsomemethod.DLL`.

La longitud máxima de un nombre de plug-in (sin incluir la extensión de archivo ni el sufijo ″64″) está limitada a 32 bytes. No existe una limitación en el número de plug-ins soportados por el servidor de bases de datos, pero la longitud máxima de la lista de plug-ins separados por comas en la configuración del gestor de bases de datos es 255 bytes. Existen dos sentencias DEFINE en el archivo de inclusión `sqlenv.h` que establecen esos dos límites:

```
#define SQL_PLUGIN_NAME_SZ     32     /* nombre del plug-in */
#define SQL_SRVCON_GSSPLUGIN_LIST_SZ 255 /* lista de plug-ins GSS API */
```

Los archivos de las bibliotecas de plug-in de seguridad tienen los permisos de archivo siguientes:

- Perteneciente al propietario de la instancia.
- Legible por todos los usuarios del sistema.
- Ejecutable por todos los usuarios del sistema.

Conceptos relacionados:

- "Configuration parameters" en la publicación *Administration Guide: Performance*
- "Plug-ins de seguridad" en la página 569
- "Ubicaciones de las bibliotecas de plug-in de seguridad" en la página 573

Tareas relacionadas:

- "Configuring DB2 with configuration parameters" en la publicación *Administration Guide: Performance*
- "Despliegue de un plug-in de recuperación de grupos" en la página 580
- "Despliegue de un plug-in de ID de usuario/contraseña" en la página 581
- "Despliegue de un plug-in de GSS-API" en la página 583
- "Despliegue de un plug-in Kerberos" en la página 584

Información relacionada:

- "Mandato UPDATE DATABASE MANAGER CONFIGURATION" en la publicación *Consulta de mandatos*
- "clnt_krb_plugin - Client Kerberos plug-in configuration parameter" en la publicación *Administration Guide: Performance*
- "clnt_pw_plugin - Client userid-password plug-in configuration parameter" en la publicación *Administration Guide: Performance*

Soporte de plug-in de seguridad para los ID de usuario de dos componentes

El producto DB2® UDB para Linux, UNIX® y Windows® da soporte al uso de los ID de usuario de dos componentes y a la correlación de esos ID de usuario de dos componentes con unos ID de autorización de dos componentes.

Por ejemplo, suponga un ID de usuario del sistema operativo Windows, formado por dos componentes: un dominio y un nombre de usuario, tal como `MEDWAY\peter`. En este ejemplo de ID de usuario de dos componentes, `MEDWAY` es un dominio, y `peter` es el nombre de usuario. En DB2, puede especificar si este ID de usuario de dos componentes se debe correlacionar con un ID de autorización de un solo componente o de dos componentes.

Antes de la aparición la Versión 8.2, en DB2 solamente podía disponer de un ID de usuario de un solo componente que se correlacionaba con un ID de autorización de un solo componente. En DB2 Versión 8.2, por omisión, los ID de usuario de un solo componente se correlacionan con unos ID de autorización de un solo componente, y los ID de usuario de dos componentes se correlacionan con unos ID de autorización de un solo componente. La correlación de un ID de usuario de dos componentes con un ID de autorización de dos componentes está soportada, pero no es el comportamiento por omisión.

Esta característica permite al usuario conectar con la base de datos utilizando:

```
db2 connect to db user MEDWAY\peter using pw
```

En este caso, si se utiliza el comportamiento por omisión, el ID de usuario MEDWAY\peter se correlacionará con el ID de autorización PETER. Si se utiliza el soporte para correlacionar un ID de usuario de dos componentes con un ID de autorización de dos componentes, en este caso el ID de autorización podría ser MEDWAY\PETER.

Para habilitar el soporte de DB2 para autenticar unos ID de usuario de dos componentes que se correlacionan con unos ID de autorización de dos componentes, debe habilitar los plug-ins de seguridad que realizan esa correlación; para ello debe definir parámetros de configuración del gestor de bases de datos. Estos parámetros de configuración se describen más abajo.

DB2 proporciona dos conjuntos de plug-ins de autenticación. Un conjunto se utiliza exclusivamente para correlacionar los ID de usuario (de uno o dos componentes) con unos ID de autorización de un solo componente. El segundo conjunto de plug-ins aporta la flexibilidad para correlacionar un ID de usuario de un solo componente con un ID de autorización de un solo componente y para correlacionar un ID de usuario de dos componentes con un ID de autorización de dos componentes.

Si un nombre de usuario de su entorno de trabajo se puede correlacionar con varias cuentas definidas en ubicaciones diferentes (tal como una cuenta local, una cuenta de dominio y cuentas de dominio fiable), entonces puede ser conveniente especificar los plug-ins que permiten realizar la correlación con los ID de autorización de dos componentes.

Es importante tener en cuenta que un ID de autorización de un solo componente, tal como PETER, y un ID de autorización de dos componentes que resulta de combinar un dominio y un ID de usuario, tal como MEDWAY\PETER, son unos ID de autorización funcionalmente distintos. El conjunto de privilegios asociados a uno de estos ID de autorización es completamente diferente del conjunto de privilegios asociados al otro ID de autorización. Debe tener cuidado al trabajar con los ID de autorización de uno y dos componentes.

La tabla siguiente muestra las clases de plug-ins proporcionadas por DB2, y los nombres de plug-in correspondientes a las implementaciones específicas de la autenticación.

Tabla 82. Plug-ins de seguridad de DB2

Tipo de autenticación	Nombre de plug-in para ID de usuario de un solo componente	Nombre de plug-in para ID de usuario de dos componentes
ID de usuario/contraseña (Cliente)	IBMOSauthclient	IBMOSauthclientTwoPart
ID de usuario/contraseña (Servidor)	IBMOSauthserver	IBMOSauthserverTwoPart
Kerberos	IBMkrb5	IBMkrb5TwoPart

Nota: En las plataformas Windows de 64 bits, se añaden los caracteres ″64″ a los nombres de plug-in mostrados aquí.

Para correlacionar un ID de usuario de dos componentes con un ID de autorización de dos componentes, debe especificar que se utilice el plug-in para ID de usuario de dos componentes, que no es el que se utiliza por omisión. Los

plug-ins de seguridad se especifican a nivel de instancia definiendo los parámetros de configuración del gestor de bases de datos referentes a la seguridad, de esta manera:

Para la autenticación de servidor que correlaciona un ID de usuario de dos componentes con un ID de autorización de dos componentes, debe asignar los valores siguientes:

- el valor `IBMOSauthserverTwoPart` a `SRVCON_PW_PLUGIN`
- el valor `IBMOSauthclientTwoPart` a `CLNT_PW_PLUGIN`

Para la autenticación de cliente que correlaciona un ID de usuario de dos componentes con un ID de autorización de dos componentes, debe asignar los valores siguientes:

- el valor `IBMOSauthserverTwoPart` a `SRVCON_PW_PLUGIN`
- el valor `IBMOSauthclientTwoPart` a `CLNT_PW_PLUGIN`

Para la autenticación Kerberos que correlaciona un ID de usuario de dos componentes con un ID de autorización de dos componentes, debe asignar los valores siguientes:

- el valor `IBMOSkrb5TwoPart` a `SRVCON_GSSPLUGIN_LIST`
- el valor `IBMkrb5TwoPart` a `CLNT_KRB_PLUGIN`

Las bibliotecas de plug-in de seguridad aceptan los ID de usuario de dos componentes que estén especificados en un formato compatible con el Gestor de cuentas de seguridad (Security Account Manager) de Microsoft® Windows. Por ejemplo, que estén en el formato: dominio\ID de usuario. Durante la conexión, los procesos de autenticación y autorización de DB2 utilizan la información especificada tanto para el dominio como para el UD de usuario.

Cuando especifica un tipo de autenticación que necesita hacer uso de un plugin de ID de usuario/contraseña o plug-in Kerberos, se utilizan por omisión los plug-ins que están listados en la columna ″Nombre de plug-in para ID de usuario de un solo componente″.

Es aconsejable implementar los plug-ins para los ID de usuario de dos componentes cuando cree nuevas bases de datos, a fin de evitar conflictos con los ID de autorización de un solo componente de las bases de datos existentes. Las nuevas bases de datos que harán uso de la autenticación basada en unos ID de autorización de dos componentes se deben crear en una instancia separada respecto de las bases de datos que hacen uso de los ID de autorización de un solo componente.

Conceptos relacionados:

- “DB2 for Windows NT and Windows NT security introduction” en la publicación *Administration Guide: Implementation*

Tareas relacionadas:

- “DB2 for Windows NT authentication with groups and domain security” en la publicación *Administration Guide: Implementation*

Información relacionada:

- “clnt_pw_plugin - Client userid-password plug-in configuration parameter” en la publicación *Administration Guide: Performance*

- "srvcon_pw_plugin - Userid-password plug-in for incoming connections at the server configuration parameter" en la publicación *Administration Guide: Performance*

Consideraciones sobre los sistemas de 32 y 64 bits para los plug-ins de seguridad

En general, una instancia de DB2® de 32 bits utiliza el plug-in de seguridad de 32 bits y una instancia de DB2 de 64 bits utiliza el plug-in de seguridad de 64 bits. Sin embargo, en una instancia de 64 bits, DB2 soporta aplicaciones de 32 bits, las cuales necesitan la biblioteca de plug-in de 32 bits.

Una instancia de base de datos en la que se pueden ejecutar tanto aplicaciones de 32 bits como de 64 bits se denomina instancia híbrida. Si dispone de una instancia híbrida y desea ejecutar aplicaciones de 32 bits, asegúrese de que los plug-ins de seguridad necesarios de 32 bits estén disponibles en el directorio de plug-ins de 32 bits. Para las instancias híbridas de DB2 en un sistema operativo UNIX®, se crean los directorios `security32` y `security64`. Para instancia híbrida de 64 bits en Windows®, los plug-ins de seguridad de 32 y 64 bits están ambos situados en el mismo directorio, pero los nombres de los plug-ins de 64 bits tienen el sufijo ″64″.

Si desea migrar desde una instancia de 32 bits a una instancia de 64 bits, debe obtener versiones de los plug-ins de seguridad que estén precompiladas para 64 bits.

Si ha adquirido los plug-ins de seguridad de un proveedor que no suministra bibliotecas de plug-in de 64 bits, puede implementar una rutina auxiliar de 64 bits que ejecute una aplicación de 32 bits. En este caso, el plug-in de seguridad es un programa externo en lugar de ser una biblioteca.

Conceptos relacionados:

- "Plug-ins de seguridad" en la página 569
- "Ubicaciones de las bibliotecas de plug-in de seguridad" en la página 573

Tareas relacionadas:

- "Migración de aplicaciones de entornos de 32 bits a entornos de 64 bits" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Determinación de problemas para plug-ins de seguridad

Los problemas producidos en los plug-ins de seguridad se notifican dos maneras: mediante errores de SQL y a través del archivo de anotaciones administrativo.

Estos son los valores de SQLCODE referentes a los plug-ins de seguridad:

- SQLCODE - 1365: se devuelve cuando se produce un error de plug-in durante la ejecución de db2start o db2stop.
- SQLCODE - 1366: se devuelve cuando existe un problema de autorización local.
- SQLCODE - 30082: se devuelve para todos los errores de plug-in referentes a la conexión.

El archivo de anotaciones administrativo es un recurso útil para depurar y administrar plug-ins de seguridad. Para ver el archivo de anotaciones administrativo en UNIX®, examine `sqllib/db2dump/<nombre de instancia>.nfy`.

Para ver el archivo de anotaciones administrativo en los sistemas operativos Windows, utilice la herramienta Visor de sucesos. Para acceder a la herramienta Visor de sucesos, seleccione el botón "Inicio" de Windows y luego seleccione `Configuración -> Panel de control -> Herramientas administrativas -l> Visor de sucesos`. Estos son los valores del archivo de anotaciones administrativo referentes a los plug-ins de seguridad:

- 13000: indica que ha fallado una llamada a una API de plug-in de seguridad GSS-API y ha devuelto un mensaje de error opcional.

  ```
  SQLT_ADMIN_GSS_API_ERROR (13000)
  El plug-in "<nombre de plug-in>" ha recibido el código de error
  "<código de error>" de la API de GSS "<nombre de api de
  gss>" y ha devuelto el mensaje de error "<mensaje de error>"
  ```

- 13001: indica que ha fallado una llamada a una API de plug-in de seguridad de DB2® y ha devuelto un mensaje de error opcional.

  ```
  SQLT_ADMIN_PLUGIN_API_ERROR(13001)
  El plug-in "<nombre de plug-in>" ha recibido el código de error
  "<código de error>" de la API de plug-in de seguridad de DB2
  "<nombre de api de gss>" y ha devuelto el mensaje de error
  "<mensaje de error>"
  ```

- 13002: indica que DB2 no ha podido cargar un plug-in.

  ```
  SQLT_ADMIN_PLUGIN_UNLOAD_ERROR (13002)
  No se pueden descargar el plug-in "<nombre de plug-in>". No es
  necesaria ninguna acción adicional.
  ```

- 13003: indica un nombre de principal incorrecto.

  ```
  SQLT_ADMIN_INVALID_PRIN_NAME (13003)
  El nombre de principal "<nombre de principal>" utilizado para
  "<nombre de plug-in>" no es válido. Corrija el nombre de
  principal.
  ```

- 13004: indica que el nombre de plug-in no es válido. Los separadores de vía de acceso ("/" en UNIX y "\" en Windows®) no se pueden utilizar como parte del nombre de plug-in.

  ```
  SQLT_ADMIN_INVALID_PLGN_NAME (13004)
  El nombre de plug-in "<nombre de plug-in>" no es válido.
  Corrija el nombre de plug-in.
  ```

- 13005: indica que no se ha podido cargar el plug-in de seguridad. Asegúrese de que el plug-in esté en el directorio correcto y que estén actualizados los parámetros de configuración apropiados del gestor de bases de datos.

  ```
  SQLT_ADMIN_PLUGIN_LOAD_ERROR (13005)
  No se puede cargar el plug-in "<nombre de plug-in>". Verifique
  que el plug-in existe y que el directorio donde reside sea correcto.
  ```

- 13006: indica que se ha producido un error inesperado en un plug-in de seguridad. Reúna toda la información proporcionada por db2support y si es posible un capture un rastreo db2trc, y luego solicite ayuda al centro de soporte de IBM®.

  ```
  SQLT_ADMIN_PLUGIN_UNEXP_ERROR (13006)
  El plug-in ha encontrado un error inesperado. Consulte al centro de
  soporte de IBM para obtener ayuda.
  ```

Nota: Si está utilizando plug-ins de seguridad en un servidor de bases de datos de un sistema Windows de 64 bits y recibe un error de carga para un plug-in de seguridad, consulte los temas Consideraciones sobre 32 y 64 bits para los conectores de seguridad y Convenios de denominación del conector de seguridad. La biblioteca de plug-in de 64 bits necesita que exista el sufijo "64" en el nombre de biblioteca, pero este sufijo no debe aparecer en los parámetros de configuración del gestor de bases de datos correspondientes al plug-in de seguridad.

Conceptos relacionados:

- "Event monitors" en la publicación *System Monitor Guide and Reference*
- "Información de error en los campos SQLCODE, SQLSTATE y SQLWARN" en la página 110
- "Variables SQLSTATE y SQLCODE en C y C++" en la página 186
- "Diferencias en SQLCODE y SQLSTATE entre sistemas de bases de datos relacionales de IBM" en la página 746
- "DB2 trace (db2trc)" en la *Guía de resolución de problemas*
- "Plug-ins de seguridad" en la página 569
- "Consideraciones sobre los sistemas de 32 y 64 bits para los plug-ins de seguridad" en la página 578
- "Las API de plug-in de seguridad" en la página 597

Información relacionada:

- "Sentencia CREATE EVENT MONITOR" en la publicación *Consulta de SQL, Volumen 2*
- "db2trc - Mandato Rastrear" en la publicación *Consulta de mandatos*
- "Mensajes de error de los plug-ins de seguridad" en la página 593
- "Las API y definiciones necesarias para los plug-ins de autenticación de GSS-API" en la página 632
- "Las API de plug-ins de recuperación de grupos" en la página 599
- "Las API del plug-in de autenticación por ID usuario/contraseña" en la página 608

Despliegue de un plug-in de recuperación de grupos

Si desea personalizar el mecanismo de recuperación de grupos del sistema de seguridad de DB2, puede desarrollar su propio plug-in de recuperación de grupos o adquirirlo de un tercero.

Una vez obtenido un plug-in de recuperación de grupos que sea apropiado para su sistema de gestión de bases de datos, puede desplegar el plug-in.

Procedimiento:

Para desplegar un plug-in de recuperación de grupos en el servidor de bases de datos, siga estos pasos:

1. Coloque la biblioteca de plug-in de recuperación de grupos en el directorio de plug-ins de grupo del servidor.
2. Actualice el parámetro de configuración *group_plugin* del gestor de bases de datos con el nombre del plug-in.

Para desplegar un plug-in de recuperación de grupos en clientes de bases de datos, siga estos pasos:

1. Coloque la biblioteca del plug-in de recuperación de grupos en el directorio del plug-in de grupo del cliente.
2. Actualice el parámetro de configuración *group_plugin* del gestor de bases de datos con el nombre del plug-in.

Conceptos relacionados:

- "Plug-ins de seguridad" en la página 569

- "Ubicaciones de las bibliotecas de plug-in de seguridad" en la página 573
- "Convenios para nombrar plug-ins de seguridad" en la página 574

Tareas relacionadas:

- "Despliegue de un plug-in de ID de usuario/contraseña" en la página 581
- "Despliegue de un plug-in de GSS-API" en la página 583
- "Despliegue de un plug-in Kerberos" en la página 584

Información relacionada:

- "group_plugin - Group plug-in configuration parameter" en la publicación *Administration Guide: Performance*

Despliegue de un plug-in de ID de usuario/contraseña

Si desea personalizar el mecanismo de autenticación basado en un ID de usuario/contraseña del sistema de seguridad de DB2, puede desarrollar sus propios plug-ins de autenticación por ID de usuario/contraseña o adquirir uno de un tercero.

Una vez obtenidos los plug-ins de autenticación por ID de usuario/contraseña que sean apropiados para su sistema de gestión de bases de datos, puede desplegar los plug-ins.

Todos los plug-ins de autenticación basados en un ID de usuario/contraseña se deben colocar en el directorio de plug-ins del cliente o del servidor, dependiendo del uso previsto de los plug-ins. Si un plug-in se coloca en el directorio de plug-ins del cliente, se utilizará para la comprobación de la autorización local y cada vez que un cliente intente conectar con el servidor. Si el plug-in se coloca en el directorio de plug-ins del servidor, se utilizará para gestionar las conexiones entrantes dirigidas al servidor y para comprobar si un ID de autorización existe y es válido cada vez que se emita una sentencia GRANT sin especificar las palabras claves USER ni GROUP.

En la mayoría de los casos, la autenticación basada en un ID de usuario/contraseña necesita solamente un plug-in en el extremo servidor. También es posible, aunque generalmente se considera menos útil, tener solamente un plug-in de ID de usuario/contraseña en el cliente. Es posible, aunque poco habitual, que sea necesario tener plug-ins compatibles de ID de usuario/contraseña tanto en el cliente como en el servidor.

Procedimiento:

Para desplegar un plug-in de autenticación por ID de usuario/contraseña en el servidor de bases de datos, siga estos pasos:

1. Coloque la biblioteca del plug-in de autenticación por ID de usuario/contraseña en el directorio de plug-ins del servidor.
2. Actualice el parámetro de configuración *srvcon_pw_plugin* del gestor de bases de datos con el nombre del plug-in del servidor.

   El servidor utiliza este plug-in para gestionar peticiones de conexión lógica (CONNECT) y conexión física (ATTACH).
3. Efectúe una de estas dos acciones:

- Asigne el tipo de autenticación CLIENT, SERVER, SERVER_ENCRYPT, DATA_ENCRYPT o DATA_ENCRYPT_CMP al parámetro de configuración *srvcon_auth* del gestor de bases de datos. O bien:
- Asigne el valor NOT_SPECIFIED al parámetro de configuración *srvcon_auth* del gestor de bases de datos y asigne el tipo de autenticación CLIENT, SERVER, SERVER_ENCRYPT, DATA_ENCRYPT o DATA_ENCRYPT_CMP al parámetro *authentication*.

Para desplegar un plug-in de autenticación por ID de usuario/contraseña en clientes de bases de datos, siga estos pasos:

1. Coloque la biblioteca del plug-in de autenticación por ID de usuario/contraseña en el directorio de plug-ins de cliente del cliente.
2. Actualice el parámetro de configuración *clnt_pw_plugin* del gestor de bases de datos con el nombre del plug-in de cliente.

   Este plug-in se carga e invoca con independencia de dónde se realice la autenticación, es decir, no solamente cuando está habilitada la autenticación del cliente.

Para la autorización local en un cliente, servidor o pasarela, que haga uso del plug-in de autenticación por ID de usuario/contraseña, siga estos pasos:

1. Coloque la biblioteca del plug-in de autenticación por ID de usuario/contraseña en el directorio de plug-ins de cliente del cliente, servidor o pasarela.
2. Actualice el parámetro de configuración *clnt_pw_plugin* del gestor de bases de datos con el nombre del plug-in.
3. Asigne el valor CLIENT, SERVER, SERVER_ENCRYPT, DATA_ENCRYPT o DATA_ENCRYPT_CMP al parámetro de configuración *authentication* del gestor de bases de datos.

Conceptos relacionados:

- "Plug-ins de seguridad" en la página 569
- "Ubicaciones de las bibliotecas de plug-in de seguridad" en la página 573

Tareas relacionadas:

- "Despliegue de un plug-in de recuperación de grupos" en la página 580
- "Despliegue de un plug-in de GSS-API" en la página 583

Información relacionada:

- "authentication - Authentication type configuration parameter" en la publicación *Administration Guide: Performance*
- "Sentencia GRANT (Autorizaciones de base de datos)" en la publicación *Consulta de SQL, Volumen 2*
- "clnt_pw_plugin - Client userid-password plug-in configuration parameter" en la publicación *Administration Guide: Performance*
- "srvcon_auth - Authentication type for incoming connections at the server configuration parameter" en la publicación *Administration Guide: Performance*
- "srvcon_pw_plugin - Userid-password plug-in for incoming connections at the server configuration parameter" en la publicación *Administration Guide: Performance*

Despliegue de un plug-in de GSS-API

Si desea personalizar el mecanismo de autenticación del sistema de seguridad de DB2, puede desarrollar sus propios plug-ins de autenticación utilizando GSS-API, o adquirir un plug-in de un tercero.

Una vez obtenidos los plug-ins de autenticación de GSS-API que sean apropiados para su sistema de gestión de bases de datos, puede desplegar los plug-ins.

En el caso de plug-ins de GSS-API o Kerberos, debe tener tipos de plug-in compatibles en el cliente y el servidor. No es necesario que los plug-ins del cliente y el servidor sean del mismo proveedor, pero deben generar y utilizar símbolos de GSS-API compatibles. Por ejemplo, cualquier combinación de plug-ins Kerberos desplegados en el cliente y el servidor es válida, pues los plug-ins Kerberos están estandarizados, en cambio, otras implementaciones menos estandarizadas de mecanismos GSS-API, tales como los certificados *x.509*, puede no ser totalmente compatible.

Todos los plug-ins de autenticación de GSS-API se deben colocar en el directorio de plug-ins del cliente o del servidor, dependiendo del uso previsto de los plug-ins. Si un plug-in se coloca en el directorio de plug-ins del cliente, se utilizará para la comprobación de la autorización local y cada vez que un cliente intente conectar con el servidor. Si el plug-in se coloca en el directorio de plug-ins del servidor, se utilizará para gestionar las conexiones entrantes dirigidas al servidor y para comprobar si un ID de autorización existe y es válido cada vez que se emita una sentencia GRANT sin especificar las palabras claves USER ni GROUP.

Procedimiento:

Para desplegar un plug-in de autenticación de GSS-API en el servidor de bases de datos, siga estos pasos:

1. Coloque la biblioteca del plug-in de autenticación de GSS-API en el directorio de plug-ins de servidor del servidor. Puede copiar varios plug-ins de GSS-API en ese directorio.
2. Actualice el parámetro de configuración *srvcon_gssplugin_list* del gestor de bases de datos con una lista ordenada, delimitada por comas, de los plug-ins instalados en el directorio de plug-ins de GSS-API.
3. Efectúe una de estas dos acciones:
   - Asigne el valor GSSPLUGIN al parámetro de configuración *srvcon_auth* del gestor de bases de datos. O bien:
   - Asigne el valor NOT_SPECIFIED al parámetro de configuración *srvcon_auth* del gestor de bases de datos y establezca *authentication* en GSSPLUGIN.

Para desplegar un plug-in de autenticación de GSS-API en clientes de bases de datos, siga estos pasos:

1. Coloque la biblioteca del plug-in de autenticación de GSS-API en el directorio de plug-ins de cliente del cliente. Puede copiar varios plug-ins de GSS-API en ese directorio. Durante la operación CONNECT/ATTACH, el cliente elige el plug-in apropiado de GSS-API para la autenticación seleccionando el primer plug-in de GSS-API contenido en la lista de plug-ins del servidor situada en el cliente.
2. Opcional: catalogue las bases de datos a las que accederá el cliente, e indique que el cliente solo aceptará como mecanismo de autenticación un plug-in de autenticación de GSS-API. Por ejemplo:

```
CATALOG DB testdb AT NODE testnode AUTHENTICATION GSSPLUGIN
```

Para la autorización local en un cliente, servidor o pasarela, que haga uso de un plug-in de autenticación de GSS-API, siga estos pasos:

1. Coloque la biblioteca del plug-in de autenticación de GSS-API en el directorio de plug-ins de cliente del cliente, servidor o pasarela.
2. Actualice el parámetro de configuración *local_gssplugin* del gestor de bases de datos con el nombre del plug-in.
3. Asigne el valor GSSPLUGIN o GSS_SERVER_ENCRYPT al parámetro de configuración *authentication* del gestor de bases de datos.

Conceptos relacionados:

- "Plug-ins de seguridad" en la página 569
- "Ubicaciones de las bibliotecas de plug-in de seguridad" en la página 573

Información relacionada:

- "authentication - Authentication type configuration parameter" en la publicación *Administration Guide: Performance*
- "Mandato CATALOG DATABASE" en la publicación *Consulta de mandatos*
- "local_gssplugin - GSS API plug-in used for local instance level authorization configuration parameter" en la publicación *Administration Guide: Performance*
- "srvcon_auth - Authentication type for incoming connections at the server configuration parameter" en la publicación *Administration Guide: Performance*
- "srvcon_gssplugin_list - List of GSS API plug-ins for incoming connections at the server configuration parameter" en la publicación *Administration Guide: Performance*
- "Ejemplos del conector de seguridad" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Despliegue de un plug-in Kerberos

Si desea personalizar el mecanismo de autenticación Kerberos del sistema de seguridad de DB2, puede desarrollar sus propios plug-ins de autenticación Kerberos o adquirir uno de un tercero.

Una vez obtenidos los plug-ins de autenticación Kerberos que sean apropiados para su sistema de gestión de bases de datos, puede desplegarlos plug-ins.

Procedimiento:

Para desplegar un plug-in de autenticación Kerberos en el servidor de bases de datos, siga estos pasos:

1. Coloque la biblioteca del plug-in de autenticación Kerberos en el directorio de plug-ins de servidor del servidor.
2. Actualice el parámetro de configuración *srvcon_gssplugin_list* del gestor de bases de datos con el nombre del plug-in Kerberos del servidor. Ese parámetro es una lista ordenada delimitada por comas. Un solo plug-in de esta lista puede ser un plug-in Kerberos.

   Si la lista está en blanco y *authentication* tiene el valor KERBEROS o KRB_SVR_ENCRYPT, se utilizará el plug-in Kerberos de DB2 por omisión: IBMkrb5.
3. Efectúe una de estas dos acciones:

- Asigne el tipo de autenticación KERBEROS o KRB_SERVER_ENCRYPT al parámetro de configuración *srvcon_auth* del gestor de bases de datos. O bien:
- Asigne el valor NOT_SPECIFIED al parámetro de configuración *srvcon_auth* del gestor de bases de datos y establezca *authentication* en el tipo de autenticación KERBEROS o KRB_SERVER_ENCRYPT.

Para desplegar un plug-in de autenticación Kerberos en clientes de bases de datos, siga estos pasos:

1. Coloque la biblioteca del plug-in de autenticación Kerberos en el directorio de plug-ins de cliente del cliente.
2. Actualice el parámetro de configuración *clnt_krb_plugin* del gestor de bases de datos con el nombre del plug-in Kerberos del cliente.

   Si *clnt_krb_plugin* está en blanco, DB2 considera que el cliente no puede utilizar la autenticación Kerberos. Esto solo es apropiado cuando el servidor no puede dar soporte a plug-ins. Consulte las limitaciones sobre el uso de plug-ins de seguridad para obtener más información. Si tanto el servidor como el cliente dan soporte a los plug-ins de seguridad, el cliente no utilizará el valor de *clnt_krb_plugin*, pues el servidor tiene listado el plug-in *IBMkrb5* de GSS-API.

   Para la autorización local en un cliente, servidor o pasarela, que haga uso de un plug-in de autenticación Kerberos, siga estos pasos:

   a. Coloque la biblioteca del plug-in de autenticación Kerberos en el directorio de plug-ins de cliente del cliente, servidor o pasarela.
   b. Actualice el parámetro de configuración *clnt_krb_plugin* del gestor de bases de datos con el nombre del plug-in.
   c. Asigne el valor KERBEROS o KRB_SERVER_ENCRYPT al parámetro de configuración *authentication* del gestor de bases de datos.

   El plug-in Kerberos proporcionado por DB2 se denomina `IBMkrb5`.
3. Opcional: catalogue las bases de datos a las que accederá el cliente, e indique que el cliente solo utilizará un plug-in de autenticación Kerberos. Por ejemplo:

```
CATALOG DB testdb AT NODE testnode AUTHENTICATION KERBEROS
        TARGET PRINCIPAL service/host@REALM
```

Nota: Para las plataformas que dan soporte a Kerberos, la biblioteca `IBMkrb5` estará presente en el directorio de plug-ins del cliente. DB2 reconoce esta biblioteca como un plug-in de GSS-API válido, pues los plug-ins Kerberos se implementan utilizando GSS-API.

Conceptos relacionados:

- "Plug-ins de seguridad" en la página 569
- "Ubicaciones de las bibliotecas de plug-in de seguridad" en la página 573

Tareas relacionadas:

- "Despliegue de un plug-in de recuperación de grupos" en la página 580
- "Despliegue de un plug-in de ID de usuario/contraseña" en la página 581
- "Despliegue de un plug-in de GSS-API" en la página 583

Información relacionada:

- "authentication - Authentication type configuration parameter" en la publicación *Administration Guide: Performance*
- "Mandato CATALOG DATABASE" en la publicación *Consulta de mandatos*

- "clnt_krb_plugin - Client Kerberos plug-in configuration parameter" en la publicación *Administration Guide: Performance*
- "srvcon_auth - Authentication type for incoming connections at the server configuration parameter" en la publicación *Administration Guide: Performance*
- "srvcon_gssplugin_list - List of GSS API plug-ins for incoming connections at the server configuration parameter" en la publicación *Administration Guide: Performance*

Capítulo 27. Desarrollo de plug-ins de seguridad

Carga de plug-ins de seguridad por DB2 587
Restricciones para bibliotecas de plug-in de seguridad 588
Códigos de retorno de los plug-ins de seguridad 590
Mensajes de error de los plug-ins de seguridad . . 593
Secuencias de llamada para las API de plug-in de seguridad 594

Carga de plug-ins de seguridad por DB2

Cada biblioteca de plug-in debe contener una función de inicialización junto con un nombre específico determinado por el tipo de plug-in:

- Plug-in de autenticación del extremo servidor: db2secServerAuthPluginInit()
- Plug-in de autenticación del extremo cliente: db2secClientAuthPluginInit()
- Plug-in de grupo: db2secGroupPluginInit()

Esta función se denomina función de inicialización del plug-in. La función de inicialización del plug-in inicializa el plug-in especificado y proporciona a DB2® información que necesita para invocar las funciones del plug-in. La función de inicialización del plug-in acepta los parámetros siguientes:

- número de versión más alto de la estructura de punteros de función que DB2 puede soportar
- puntero que apunta a una estructura que contiene los punteros que apuntan a todas las API que necesitan implementación
- puntero que apunta a una función que añade mensajes de registro al archivo db2diag.log
- puntero que apunta a una serie de caracteres representativa de un mensaje de error
- longitud del mensaje de error

Lo siguiente es una signatura de función para la función de inicialización de un plug-in de recuperación de grupos:

```
SQL_API_RC SQL_API_FN db2secGroupPluginInit(
 db2int32 version,
 void *group_fns,
  db2secLogMessage *logMessage_fn,
  char **errormsg,
 db2int32  *errormsglen);
```

Nota: Las bibliotecas de plug-in solo se pueden implementar en C o C++. Si la biblioteca de plug-in se compila como C++, todas las funciones se deben declarar con: `extern "C"`. DB2 depende del cargador dinámico subyacente del sistema operativo para manejar los métodos constructores y destructores de C++ utilizados dentro de una biblioteca de plug-in de C++ escrita por el usuario.

Esta es la única función de la biblioteca de plug-in que debe tener un nombre de función prescrito. Las demás funciones de plug-in se referencian mediante punteros de función devueltos por la función de inicialización. Los plug-ins de servidor se cargan en el servidor cuando se ejecuta db2start. Los plug-ins de cliente se cargan en el cliente cuando sea necesario. Inmediatamente después de cargar la biblioteca de plug-in, DB2 determina la ubicación de la función y luego la invoca. La tarea específica de esta función es la siguiente:

- convertir el puntero de función en un puntero que apunta a un estructura de funciones apropiada
- especificar los punteros que apuntan a las demás funciones de la biblioteca
- especificar el número de versión de la estructura de punteros de función devuelta

DB2 puede potencialmente invocar la función de inicialización de plug-in más de una vez. Un caso en el que esto ocurre es cuando una aplicación carga dinámicamente la biblioteca del cliente DB2, la descarga y la vuelve a cargar, y luego ejecuta funciones de autenticación desde un plug-in tanto antes como después de la recarga. En este caso, la biblioteca de plug-in podría no descargarse y luego cargarse de nuevo, pero este comportamiento varía según el sistema operativo.

Otro caso en el que DB2 emite varias llamadas a una función de inicialización de plug-in es cuando se ejecutan procedimientos almacenados o llamadas de sistema federado, donde el propio servidor de bases de datos puede actuar como cliente. Si los plug-ins de cliente y servidor contenidos en el servidor de bases de datos residen en el mismo archivo, DB2 podría invocar dos veces la función de inicialización de plug-in.

Si el plug-in detecta que `db2secGroupPluginInit` se invoca más de una vez, debe interpretar esto como si le indicara que debe finalizar y reinicializar la biblioteca de plug-in. De esta manera, la función de inicialización de plug-in debe realizar el proceso completo de limpieza que realizaría una llamada a `db2secPluginTerm` antes de devolver de nuevo el conjunto de punteros de función.

En un servidor DB2 que se ejecute en un sistema operativo UNIX®, DB2 puede potencialmente cargar e inicializar bibliotecas de plug-in más de una vez después de la ejecución de db2start en procesos diferentes.

Conceptos relacionados:

- "Plug-ins de seguridad" en la página 569
- "Ubicaciones de las bibliotecas de plug-in de seguridad" en la página 573

Información relacionada:

- "Restricciones para bibliotecas de plug-in de seguridad" en la página 588
- "Códigos de retorno de los plug-ins de seguridad" en la página 590
- "Secuencias de llamada para las API de plug-in de seguridad" en la página 594
- "db2secGroupPluginInit - Inicializar plug-in de grupo" en la página 601
- "db2secPluginTerm - Depurar recursos de plug-in de grupo" en la página 602
- "db2secClientAuthPluginInit - Inicializar el plug-in de autenticación del cliente" en la página 615
- "db2secServerAuthPluginInit - Inicializar el plug-in de autenticación del servidor" en la página 627

Restricciones para bibliotecas de plug-in de seguridad

A continuación se indican las restricciones existentes en el desarrollo de bibliotecas de plug-in.

C-linkage

Las bibliotecas de plug-in se deben enlazar con C-linkage. Los archivos de

cabecera donde residen los prototipos, las estructuras de datos necesarias para implementar los plug-ins, y las definiciones de códigos de error se proporcionan solo para C/C++. Las funciones que DB2 resuelve durante la carga se deben declarar con ″C″ externo si la biblioteca de plug-in se compila como C++.

El CLR (common language runtime) .NET no está soportado
El CLR (common language runtime) .NET no está soportado para compilar y enlazar el código fuente de bibliotecas de plug-in.

Manejadores de señales
Las bibliotecas de plug-in no deben instalar manejadores de señales ni cambiar la máscara de señales, pues el hacerlo interferiría con los manejadores de señales de DB2. El interferir con los manejadores de señales de DB2 podría interferir gravemente con la capacidad de DB2 para notificar y corregir errores, incluidas las interrupciones de programa contenidas en el propio código del plug-in. Las bibliotecas de plug-in no deben tampoco emitir nunca excepciones de C++, pues ello puede también interferir con el manejo de errores realizado por DB2.

Hebras protegidas
Las bibliotecas de plug-in deben tener hebras protegidas y ser reentrantes. La función de inicialización de plug-in es la única API que no es necesario que sea reentrante. Esto es debido a que la función de inicialización de plug-in puede potencialmente ser invocada varias veces desde procesos diferentes. En este caso, el plug-in libera todos los recursos utilizados y se reinicializa a sí mismo.

Manejadores de rutinas de salida y cancelación de las llamadas de biblioteca C estándar y de sistema operativo
Las bibliotecas de plug-in no deben cancelar las llamadas de biblioteca C estándar ni de sistema operativo. Las bibliotecas de plug-in no deben tampoco instalarse en manejadores de rutinas de salida ni manejadores `pthread_atfork`. El uso de manejadores de rutinas de salida no es recomendable, pues pueden descargarse antes de que finalice el programa.

Dependencias de biblioteca
En Linux o Unix, los procesos por los que se cargan las bibliotecas de plug-in pueden ser setuid o setgid, por lo que no pueden contar con las variables de entorno `$LD_LIBRARY_PATH`, `$SHLIB_PATH` o `$LIBPATH` para encontrar bibliotecas dependientes. Por tanto, las bibliotecas de plug-in no deben depender de otras bibliotecas, a menos que las bibliotecas dependientes sean accesibles a través de otros métodos, tales como los siguientes:

- residir en `/lib` o `/usr/lib`
- los directorios donde residen estar definidos a nivel del sistema operativo (tal como ocurre en el archivo `ld.so.conf` en Linux)
- estar especificadas en la variable RPATH en la propia biblioteca de plug-in

Esta restricción no es aplicable a los sistemas operativos Windows.

Colisiones de símbolos
Cuando sea posible, las bibliotecas de plug-in se deben compilar y enlazar utilizando las opciones disponibles que reduzcan la probabilidad de colisiones de símbolos, tales como las opciones que reducen las referencias simbólicas externas no vinculadas. Por ejemplo, utilice la opción de enlazador ″-Bsymbolic″ en HP, Sun Solaris y Linux para ayudar a prevenir los problemas referentes a la colisión de símbolos. Sin embargo, para un

plug-in escrito en la plataforma AIX, no utilice la opción de enlazador "-brtl" explícitamente ni implícitamente, pues ello causaría un problema.

Aplicaciones de 32 y 64 bits
Las aplicaciones de 32 bits deben utilizar plug-ins de 32 bits. Las aplicaciones de 64 bits deben utilizar plug-ins de 64 bits. Consulte el tema Consideraciones sobre 32 y 64 bits para los conectores de seguridad para obtener más detalles.

Cadenas de texto
No hay seguridad de que las cadenas de texto de entrada terminen en nulo, y no es obligatorio que las cadenas de salida terminen en nulo. En lugar de eso, se asignan longitudes enteras a todas las cadenas de entrada, y se asignan punteros a enteros para las longitudes a devolver.

Paso de parámetros authid
El parámetro authid que DB2 pasa a un plug-in (parámetro de entrada authid) contiene un ID de autorización escrito en mayúsculas, con los blancos de relleno eliminados. El parámetro authid que un plug-in devuelve a DB2 (parámetro authid de salida) no necesita ningún tratamiento especial, pero DB2 lo convierte a mayúsculas y lo rellena con blancos de acuerdo con el estándar DB2 interno.

Límites de tamaño para parámetros
Las API de plug-in utilizan lo siguiente como límites de longitud para los parámetros:

```
#define DB2SEC_MAX_AUTHID_LENGTH 255
#define DB2SEC_MAX_USERID_LENGTH 255
#define DB2SEC_MAX_USERNAMESPACE_LENGTH 255
#define DB2SEC_MAX_PASSWORD_LENGTH 255
#define DB2SEC_MAX_DBNAME_LENGTH 128
```

Una implementación de plug-in determinada puede necesitar o aplicar longitudes máximas menores para los ID de autorización, los ID de usuario y las contraseñas. En particular, los plug-in de autenticación del sistema operativo proporcionados con DB2 UDB se ajustan a los límites máximos de longitud aplicados por el sistema operativo para el nombre de usuario, de grupo y de espacio de nombres en los casos en los que esos límites sean más restrictivos que los indicados más arriba.

Conceptos relacionados:

- "Plug-ins de seguridad" en la página 569
- "Ubicaciones de las bibliotecas de plug-in de seguridad" en la página 573

Códigos de retorno de los plug-ins de seguridad

Todas las API de plug-in de seguridad deben devolver un valor entero para indicar si la API se ha ejecutado con éxito o no. El valor de código de retorno 0 indica que la API se ejecutó satisfactoriamente. Todos los códigos de retorno negativos, con la excepción de -3, -4 y -5, indican que la API encontró un error.

Todos los códigos de retorno negativos devueltos por las API de plug-in de seguridad se correlacionan con los códigos de SQL -1365, -1366 o -30082, excepto los códigos de retorno -3, -4 o -5. Los valores -3, -4 y -5 se utilizan para indicar si un ID de autorización (AUTHID) representa o no un usuario o grupo válido.

Todos los códigos de retorno de las API de plug-in de seguridad están definidos en `db2secPlugin.h`, que reside en el directorio include de DB2: `SQLLIB/include`.

La tabla siguiente proporciona detalles sobre todos los códigos de retorno de los plug-ins de seguridad:

Tabla 83. Códigos de retorno de plug-ins de seguridad

Código de retorno	Valor de sentencia define	Significado	API aplicable
0	DB2SEC_PLUGIN_OK	La API de plug-in se ejecutó satisfactoriamente.	Todas
-1	DB2SEC_PLUGIN_UNKNOWNERROR	La API de plug-in encontró un error inesperado.	Todas
-2	DB2SEC_PLUGIN_BADUSER	El ID de usuario pasado como entrada no está definido.	db2secGenerateInitialCred db2secValidatePassword db2secRemapUserid db2secGetGroupsForUser
-3	DB2SEC_PLUGIN_INVALIDUSERORGROUP	No existe el usuario o grupo especificado.	db2secDoesAuthIDExist db2secDoesGroupExist
-4	DB2SEC_PLUGIN_USERSTATUSNOTKNOWN	Estado de usuario desconocido. DB2 no trata esta condición como un error. Es utilizada por una sentencia GRANT para determinar si un ID de autorización representa un usuario o grupo de sistema operativo.	db2secDoesAuthIDExist
-5	DB2SEC_PLUGIN_GROUPSTATUSNOTKNOWN	Estado de grupo desconocido. DB2 no trata esta condición como un error. Es utilizada por una sentencia GRANT para determinar si un ID de autorización representa un usuario o grupo de sistema operativo.	db2secDoesGroupExist
-6	DB2SEC_PLUGIN_UID_EXPIRED	El ID de usuario ha caducado.	db2secValidatePassword db2GetGroupsForUser db2secGenerateInitialCred
-7	DB2SEC_PLUGIN_PWD_EXPIRED	La contraseña ha caducado.	db2secValidatePassword db2GetGroupsForUser db2secGenerateInitialCred
-8	DB2SEC_PLUGIN_USER_REVOKED	Se ha revocado al usuario.	db2secValidatePassword db2GetGroupsForUser
-9	DB2SEC_PLUGIN_USER_SUSPENDED	Se ha suspendido al usuario.	db2secValidatePassword db2GetGroupsForUser
-10	DB2SEC_PLUGIN_BADPWD	Contraseña incorrecta.	db2secValidatePassword db2secRemapUserid db2secGenerateInitialCred
-11	DB2SEC_PLUGIN_BAD_NEWPASSWORD	Contraseña nueva incorrecta.	db2secValidatePassword db2secRemapUserid
-12	DB2SEC_PLUGIN_CHANGEPASSWORD _NOTSUPPORTED	Cambio de contraseña no soportado.	db2secValidatePassword db2secRemapUserid db2secGenerateInitialCred
-13	DB2SEC_PLUGIN_NOMEM	El plug-in no ha podido asignar memoria debido a que no existe memoria suficiente.	Todas
-14	DB2SEC_PLUGIN_DISKERROR	El plug-in ha encontrado un error de disco.	Todas

Tabla 83. Códigos de retorno de plug-ins de seguridad (continuación)

Código de retorno	Valor de sentencia define	Significado	API aplicable
-15	DB2SEC_PLUGIN_NOPERM	El plug-in no ha podido acceder a un archivo debido a que no tiene los permisos correctos para el archivo.	Todas
-16	DB2SEC_PLUGIN_NETWORKERROR	El plug-in ha encontrado un error de red.	Todas
-17	DB2SEC_PLUGIN_CANTLOADLIBRARY	El plug-in no puede cargar una biblioteca necesaria.	db2secGroupPluginInit db2secClientAuthPluginInit db2secServerAuthPluginInit
-18	DB2SEC_PLUGIN_CANT_OPEN_FILE	El plug-in no puede abrir y leer un archivo por una razón que no es la ausencia de un archivo ni la falta de permisos adecuados para el archivo.	Todas
-19	DB2SEC_PLUGIN_FILENOTFOUND	El plug-in no puede abrir y leer un archivo porque el archivo falta en el sistema de archivos.	Todas
-20	DB2SEC_PLUGIN_CONNECTION_DISALLOWED	El plug-in está rechazando la conexión debido a la restricción que establece a qué base de datos se puede conectar el plug-in o qué dirección TCP/IP puede conectar con una base de datos determinada.	Todas las API de plug-in del extremo servidor.
-21	DB2SEC_PLUGIN_NO_CRED	Plug-in de la API de GSS solamente: falta la credencial inicial del cliente.	db2secGetDefaultLoginContext db2secServerAuthPluginInit
-22	DB2SEC_PLUGIN_CRED_EXPIRED	Plug-in de la API de GSS solamente: la credencial del cliente ha caducado.	db2secGetDefaultLoginContext db2secServerAuthPluginInit
-23	DB2SEC_PLUGIN_BAD_PRINCIPAL_NAME	Plug-in de la API de GSS solamente: el nombre de principal no es válido.	db2secProcessServerPrincipalName
-24	DB2SEC_PLUGIN_NO_CON_DETAILS	Este código de retorno es devuelto por la llamada `db2secGetConDetails` (por ejemplo, desde DB2 al plug-in) para indicar que DB2 no puede determinar la dirección TCP/IP del cliente.	db2secGetConDetails
-25	DB2SEC_PLUGIN_BAD_INPUT_PARAMETERS	Algunos parámetros no son válidos o faltan cuando se invoca la API de plug-in.	Todas
-26	DB2SEC_PLUGIN_INCOMPATIBLE_VER	La versión de las API notificada por el plug-in no es compatible con DB2.	db2secGroupPluginInit db2secClientAuthPluginInit db2secServerAuthPluginInit
-27	DB2SEC_PLUGIN_PROCESS_LIMIT	El plug-in se quedó sin recursos al intentar crear un nuevo proceso.	Todas
-28	DB2SEC_PLUGIN_NO_LICENSES	El plug-in ha encontrado un problema de licencia de usuario. Existe la posibilidad de que la licencia del mecanismo subyacente haya alcanzado su límite.	Todas

Conceptos relacionados:

- "Plug-ins de seguridad" en la página 569
- "Determinación de problemas para plug-ins de seguridad" en la página 578

Mensajes de error de los plug-ins de seguridad

Cuando se produce un error en una API de plug-in de seguridad, la API puede devolver una cadena de texto ASCII en el campo `errormsg` para proporcionar una descripción del problema más específica que el código de retorno. Por ejemplo, la cadena de texto proporcionada en `errormsg` puede contener lo siguiente: `"El archivo /home/db2inst1/mypasswd.txt no existe".` DB2 escribe la cadena de texto completa en el archivo de notificaciones de administración de DB2 y también incluye una versión abreviada como símbolo en algunos mensajes de SQL. Debido a que los símbolos contenidos en los mensajes de SQL solo pueden tener una longitud limitada, es recomendable que esos mensajes se mantengan cortos y que las partes variables importantes de esos mensajes aparezcan al principio de la cadena de texto. Para facilitar la depuración de errores, puede añadir el nombre del plug-in de seguridad al mensaje de error.

Para los errores no urgentes, tales como los errores de contraseña caducada, la cadena de texto contenida en `errormsg` solo se descargará a un archivo cuando el parámetro de configuración DIAGLEVEL del gestor de bases de datos tenga el valor 4.

El plug-in de seguridad es el encargado de asignar la memoria para estos mensajes de error. Por tanto, el plug-in debe también proporcionar una API para liberar esta memoria: `db2secFreeErrormsg`.

DB2 solo examina el campo `errormsg` si una API devuelve un valor distinto de cero. Por tanto, el plug-in no debe asignar memoria para el mensaje de error devuelto si no existe ningún error.

Durante la inicialización, se pasa un puntero de función para el registro de mensajes, `logMessage_fn`, a los plug-ins de grupo, de cliente y de servidor. Los plug-ins pueden utilizar la función para registrar en db2diag.log la información de depuración disponible. Por ejemplo:

```
// Registrar un mensaje para indicar la inicialización satisfactoria
(*(logMessage_fn))(DB2SEC_LOG_CRITICAL,
                   "db2secGroupPluginInit successful",
                    strlen("db2secGroupPluginInit successful"));
```

Para obtener más detalles sobre cada parámetro de la función db2secLogMessage, consulte la API de inicialización correspondiente a cada tipo de plug-in.

Conceptos relacionados:

- "Plug-ins de seguridad" en la página 569
- "Determinación de problemas para plug-ins de seguridad" en la página 578
- "Las API de plug-in de seguridad" en la página 597
- "Soporte de plug-in de seguridad para los ID de usuario de dos componentes" en la página 575

Información relacionada:

- "Códigos de retorno de los plug-ins de seguridad" en la página 590

Secuencias de llamada para las API de plug-in de seguridad

Existen cinco situaciones principales en las que DB2 invoca las API de plug-in de seguridad:

- Desde un cliente para una conexión de base de datos.
- Desde un cliente, servidor o pasarela para una autorización local.
- Desde un servidor para una conexión de base de datos.
- Desde un servidor para una sentencia GRANT.
- Desde un servidor para obtener una lista de los grupos a los que pertenece un ID de autorización.

Nota: El servidor DB2 trata como aplicaciones cliente las acciones sobre bases de datos que requieren autorización local, tales como db2start, db2stop y db2trc.

Para cada una de estas operaciones, la secuencia utilizada por DB2 para llamar a las API de plug-in de seguridad es diferente y apropiada para cada caso. Estas son las secuencias de las API invocadas por DB2 para cada una de las situaciones posibles.

Desde un cliente para una conexión de base de datos

Cuando el tipo de autenticación configurado por el usuario es CLIENT, la aplicación cliente DB2 invoca las siguientes API de plug-in de seguridad:

- `db2secGetDefaultLoginContext();`
- `db2secValidatePassword();`
- `db2secFreetoken();`

En el caso de una autenticación implícita, es decir, cuando el usuario se conecta sin especificar un ID de usuario ni contraseña determinados, se invoca la API `db2secValidatePassword` si el usuario está utilizando un plug-in de ID de usuario/contraseña. Esta API permite al desarrollador del plug-in prohibir la autenticación implícita si es necesario.

En una autenticación implícita, si el parámetro de configuración *authentication* del gestor de bases de datos tiene un valor distinto de CLIENT (lo que implica que la autenticación se realiza en el servidor), la aplicación invoca las siguientes API de plug-in de seguridad para el mecanismo de autenticación basado en un ID de usuario/contraseña:

- `db2secGetDefaultLoginContext();`
- `db2secFreeToken();`

En una autenticación implícita, si el parámetro *authentication* tiene un valor distinto de CLIENT (lo que implica que la autenticación se realiza en el servidor), la aplicación invoca las siguientes API de plug-in de seguridad para los plug-ins de GSS-API. (La llamada a `gss_init_sec_context()` utiliza GSS_C_NO_CREDENTIAL como credencial de entrada.)

- `db2secGetDefaultLoginContext();`
- `db2secProcessServerPrincipalName();`
- `gss_init_sec_context();`
- `gss_release_buffer();`
- `gss_release_name();`
- `gss_delete_sec_context();`
- `db2secFreeToken();`

La API *gss_init_sec_context()* se puede llamar dos veces si el servidor devuelve un símbolo de autenticación mutua.

En una autenticación explícita, si *authentication* tiene el valor CLIENT, la aplicación cliente DB2 invoca las siguientes API de plug-in de seguridad:

- `db2secRemapUserid();`
- `db2secValidatePassword();`
- `db2secFreeToken();`

En una autenticación explícita, si *authentication* tiene un valor distinto de CLIENT, la aplicación invoca las siguientes API de plug-in de seguridad para el mecanismo de autenticación basado en un ID de usuario/contraseña:

- `db2secRemapUserid();`

Si el tipo de autenticación negociado es GSS-API o Kerberos, la aplicación cliente invoca las siguientes API de plug-in de seguridad para los plug-ins de GSS-API, según la secuencia mostrada. Estas API se utilizan para la autenticación implícita o explícita (conexión a una base de datos en la que se especifica tanto el ID de usuario como la contraseña) a menos que se indique otra cosa.

- `db2secProcessServerPrincipalName();`
- `db2secGenerateInitialCred();` (para la autenticación explícita solamente)
- `gss_init_sec_context();`
- `gss_release_buffer ();`
- `gss_release_name();`
- `gss_release_cred();`
- `db2secFreeInitInfo();`
- `gss_delete_sec_context();`
- `db2secFreeToken();`

La API gss_init_sec_context() se puede llamar dos veces si el servidor devuelve un símbolo de autenticación mutua.

Desde un cliente, servidor o pasarela para una autorización local

Para una autorización local, el mandato de DB2 utilizado invocará las siguientes API de plug-in de seguridad:

- `db2secGetDefaultLoginContext();`
- `db2secGetGroupsForUser();`
- `db2secFreeToken();`
- `db2secFreeGroupList();`

Estas API se invocan para los mecanismos de autenticación basados en un ID de usuario/contraseña y para los mecanismos de autenticación basados en GSS-API.

Desde un servidor para una conexión de base de datos

Para una conexión de base de datos desde el servidor de bases de datos, el proceso agente o hebra de DB2 invoca las siguientes API de plug-in de seguridad para el mecanismo de autenticación basado en un ID de usuario/contraseña:

- `db2secValidatePassword();` (solamente si *authentication* no es CLIENT)

- `db2secGetAuthIDs();`
- `db2secGetGroupsForUser();`
- `db2secFreeToken();`
- `db2secFreeGroupList();`

Para una conexión de base de datos desde el servidor de bases de datos, el proceso agente o hebra de DB2 invoca las siguientes API de plug-in de seguridad para el mecanismo de autenticación basado en GSS-API:

- `gss_accept_sec_context();`
- `gss_release_buffer();`
- `db2secGetAuthIDs();`
- `db2secGetGroupsForUser();`
- `gss_delete_sec_context();`
- `db2secFreeToken();`
- `db2secFreeGroupList();`

Desde un servidor para una sentencia GRANT

Para una sentencia GRANT que no especifica la palabra clave USER ni GROUP (por ejemplo, "`GRANT CONNECT ON DATABASE TO user1`"), DB2 debe poder determinar si `user1` es un usuario, un grupo o ambas cosas. Por tanto, DB2 invocará las siguientes API de plug-in de seguridad:

- `db2secDoesGroupExist();`
- `db2secDoesAuthIDExist();`

Desde un servidor para obtener una lista de los grupos a los que pertenece un ID de autorización

Desde el servidor de bases de datos, cuando el usuario necesita obtener una lista de los grupos a los que pertenece un ID de autorización, DB2 invoca la siguiente API de plug-in de seguridad utilizando solamente el ID de autorización como dato de entrada:

- `db2secGetGroupsForUser();`

No se obtendrán símbolos procedentes de otros plug-ins de seguridad.

Conceptos relacionados:

- "Plug-ins de seguridad" en la página 569
- "Las API de plug-in de seguridad" en la página 597

Capítulo 28. Las API de plug-in de seguridad

Las API de plug-in de seguridad 597
Las API de plug-in de grupo 599
Las API de plug-ins de recuperación de grupos 599
db2secGroupPluginInit - Inicializar plug-in de grupo 601
db2secPluginTerm - Depurar recursos de plug-in de grupo 602
db2secGetGroupsForUser - Obtener la lista de grupos del usuario 602
db2secDoesGroupExist - Comprobar si existe el grupo 606
db2secFreeGroupListMemory - Liberar memoria de lista de grupo 607
db2secFreeErrormsg - Liberar la memoria de mensajes de error 608
Las API de plug-in de autenticación de usuario 608
Las API del plug-in de autenticación por ID usuario/contraseña 608
db2secClientAuthPluginInit - Inicializar el plug-in de autenticación del cliente 615
db2secClientAuthPluginTerm - Depurar recursos de plug-in de autenticación de cliente 616
db2secRemapUserid - Reasignar ID de usuario y contraseña 617
db2secGetDefaultLoginContext - Obtener contexto de conexión por omisión 618
db2secGenerateInitialCred - Generar credenciales iniciales 620
db2secValidatePassword - Validar contraseña 622
db2secProcessServerPrincipalName - Procesar nombre de principal de servicio devuelto desde servidor 624
db2secFreeToken - Liberar memoria retenida por símbolo (token). 625
db2secFreeInitInfo - Liberar recursos retenidos por db2secGenerateInitialCred() 626
db2secServerAuthPluginInit - Inicializar el plug-in de autenticación del servidor 627
db2secServerAuthPluginTerm - Depurar recursos de plug-in de autenticación de servidor 629
db2secGetAuthIDs - Obtener los ID de autenticación 629
db2secDoesAuthIDExist - Comprobar si existe el ID de autenticación 631
Las API de plug-in de GSS-API 632
Las API y definiciones necesarias para los plug-ins de autenticación de GSS-API 632
Restricciones para los plug-in de autenticación de GSS-API 633
Creación de versiones de la API de plug-in de seguridad 634

Las API de plug-in de seguridad

Para que pueda personalizar el mecanismo de autorización de DB2®, DB2 proporciona varias API que puede utilizar para modificar plug-ins existentes o crear nuevos plug-ins de seguridad.

Cuando desarrolla un plug-in de seguridad, es necesario que implemente las funciones de autenticación estándar que serán invocadas por DB2. Para los tres tipos de plug-ins disponibles, la funcionalidad que necesita implementar es la siguiente:

Recuperación de grupos
Recupera información sobre pertenencia a grupos para un usuario determinado y determina si una cadena de caracteres dada representa un nombre de grupo válido.

Autenticación por ID de usuario/contraseña
Autenticación que identifica el contexto de seguridad por omisión (cliente solamente), valida y opcionalmente cambia una contraseña, determina si una cadena de caracteres proporcionada representa un usuario válido (servidor solamente), modifica el ID de usuario o contraseña proporcionados en el cliente antes de enviarlos al servidor (cliente solamente) y devuelve el ID de autorización de DB2 asociado a un usuario determinado.

Autenticación GSS-API
Autenticación que implementa las funciones de GSS-API necesarias,

identifica el contexto de seguridad por omisión (extremo cliente solamente), genera las credenciales iniciales basándose en un ID de usuario y contraseña y opcionalmente cambia la contraseña (extremo cliente solamente), crea y acepta certificados de seguridad, y devuelve el ID de autorización de DB2 asociado a un contexto de seguridad determinado de GSS-API.

Las definiciones siguientes describen la terminología utilizada en las descripciones de las API de plug-in.

Plug-in
Biblioteca de carga dinámica que DB2 carga para acceder a funciones de autenticación escritas por el usuario.

Autenticación implícita
Conexión a una base de datos sin especificar un ID de usuario ni una contraseña.

Autenticación explícita
Conexión a una base de datos en la que se especifican tanto el ID de usuario como la contraseña.

ID de autorización
ID interno que representa a un individuo o grupo al cual se otorgan autorizaciones y privilegios dentro de la base de datos. Internamente, el ID de autorización de DB2 tiene un mínimo de 8 caracteres, escritos en mayúsculas (con blancos de relleno hasta completar los 8 caracteres). Actualmente, DB2 necesita que los ID de autorización, los ID de usuario, las contraseñas, los nombres de grupo, los espacios de nombres y los nombres de dominio se puedan representar utilizando el juego de caracteres ASCII de 7 bits. La longitud máxima de un ID de autorización es 30 caracteres.

Autorización local
Autorización que es local respecto del servidor o cliente que realiza la implementación de la autorización, y que comprueba si un usuario está autorizado para ejecutar una acción, que no sea la de conectar con la base de datos mediante una autorización, tal como iniciar y detener el gestor de bases de datos, activar y desactivar la función de rastreo de DB2 o actualizar la configuración del gestor de bases de datos.

Espacio de nombres
Colección o agrupación de usuarios en la que los identificadores de usuarios individuales deben ser exclusivos. Son ejemplos típicos de ello los dominios de Windows® y las regiones de Kerberos. Por ejemplo, dentro del dominio de Windows "usa.company.com", todos los nombres de usuario deben ser exclusivos. Por ejemplo, "user1@usa.company.com". Sin embargo, un mismo ID de usuario utilizado en otro dominio, tal como "user1@canada.company.com", hace referencia a una persona diferente. Un identificador de usuario totalmente calificado incluye un ID de usuario y un espacio de nombres; por ejemplo, "usuario@nombre.dominio" o "dominio\usuario".

Conceptos relacionados:

- "Plug-ins de seguridad" en la página 569

Las API de plug-in de grupo

Las API de plug-ins de recuperación de grupos

Para la biblioteca de plug-in de recuperación de grupos, debe implementar las API siguientes:

```
SQL_API_RC SQL_API_FN db2secGroupPluginInit(
 db2int32 version,
 void *group_fns,
 db2secLogMessage *logMessage_fn,
 char **errormsg,
 db2int32  *errormsglen);
```

Nota: La función mostrada más arriba utiliza como entrada un puntero que apunta a una función, `*logMessage_fn`, con el prototipo siguiente:

```
SQL_API_RC (SQL_API_FN db2secLogMessage) (
  db2int32 level,
 void *data,
 db2int32 length);
```

```
SQL_API_RC SQL_API_FN db2secPluginTerm(char **errormsg,
      db2int32  *errormsglen);
```

```
SQL_API_RC SQL_API_FN db2secGetGroupsForUser(
 const char *authid,
 db2int32  authidlen,
 const char *userid,
 db2int32  useridlen,
 const char *usernamespace,
 db2int32  usernamespacelen,
 db2int32  usernamespacetype,
 const char *dbname,
 db2int32  dbnamelen,
 const void *token,
 db2int32 tokentype,
 db2int32 location,
 const char *authpluginname,
 db2int32  authpluginname,
 char **grouplist,
 db2int32  *numgroups,
 char **errormsg,
 db2int32  *errormsglen);
```

```
SQL_API_RC SQL_API_FN db2secDoesGroupExist(
  const char *groupname,
  db2int32  groupnamelen
  char **errormsg,
  db2int32  *errormsglen);
```

```
SQL_API_RC SQL_API_FN db2secFreeGroupListMemory(
  char *ptr,
  char **errormsg,
  db2int32  *errormsglen);
```

```
SQL_API_RC SQL_API_FN db2secFreeErrormsg(char *msgtobefree);
```

La única API que se debe poder resolver externamente es `db2secGroupPluginInit()`. Esta función utiliza `void *` como parámetro de entrada, que se debe convertir al tipo:

```
typedef struct db2secGroupFunctions_1
{
  db2int32 version;
  db2int32 plugintype;
  SQL_API_RC (SQL_API_FN * db2secGetGroupsForUser) (
    const char *authid,
```

```
    db2int32  authidlen,
    const char *userid,
    db2int32  useridlen,
    const char *usernamespace,
    db2int32  usernamespacelen,
    db2int32  usernamespacetype,
    const char *dbname,
    db2int32  dbnamelen,
    const void *token,
    db2int32 tokentype,
    db2int32 location,
    const char *authpluginname,
    db2int32  authpluginnamelen,
    void **grouplist,
    db2int32  *numgroups,
    char **errormsg,
    db2int32  *errormsglen);

 SQL_API_RC (SQL_API_FN * db2secDoesGroupExist)(
    const char *groupname,
    db2int32  groupnamelen,
    char **errormsg,
    db2int32  *errormsglen);

  SQL_API_RC (SQL_API_FN * db2secFreeGroupListMemory)(
    void *ptr,
    char **errormsg,
    db2int32  *errormsglen);

  SQL_API_RC (SQL_API_FN * db2secFreeErrormsg)(
    char *msgtobefree);

 SQL_API_RC (SQL_API_FN * db2secPluginTerm)(
    char **errormsg,
    db2int32  *errormsglen);

} db2secGroupFunctions_1;
```

`db2secGroupPluginInit()` asignará las direcciones para el resto de las funciones disponibles externamente.

Nota: El sufijo _1 indica que la estructura corresponde a la versión 1 de la API. Las versiones de interfaz subsiguientes tendrán la extensión _2, _3 y así sucesivamente.

Conceptos relacionados:

- "Plug-ins de seguridad" en la página 569
- "Las API de plug-in de seguridad" en la página 597

Tareas relacionadas:

- "Despliegue de un plug-in de recuperación de grupos" en la página 580

Información relacionada:

- "db2secGroupPluginInit - Inicializar plug-in de grupo" en la página 601
- "db2secPluginTerm - Depurar recursos de plug-in de grupo" en la página 602
- "db2secGetGroupsForUser - Obtener la lista de grupos del usuario" en la página 602
- "db2secDoesGroupExist - Comprobar si existe el grupo" en la página 606
- "db2secFreeGroupListMemory - Liberar memoria de lista de grupo" en la página 607

- "db2secFreeErrormsg - Liberar la memoria de mensajes de error" en la página 608

db2secGroupPluginInit - Inicializar plug-in de grupo

Esta es la función de inicialización correspondiente a la biblioteca de plug-in que DB2 invoca inmediatamente después de cargar la biblioteca. El puntero `functions` se debe convertir a la estructura `group_functions` apropiada para la versión de interfaz.

Sintaxis de la API de C:

```
SQL_API_RC SQL_API_FN db2secGroupPluginInit(
  db2int32 version,
  void *group_fns,
  db2secLogMessage *logMessage_fn,
  char **errormsg,
  db2int32  *errormsglen);
```

Entrada:

db2int32 version
Es el número de versión más alto de la API al que DB2 da soporte actualmente.

*db2secLogMessage *logMessage_fn*
Es un puntero que apunta a una función proporcionada por DB2. El plug-in puede invocar esta función para registrar en `db2diag.log` más información sobre errores para utilizarla con fines de depuración o información. El primer parámetro debe utilizar la sentencia DEFINE contenida en `db2secPlugin.h` y los dos últimos parámetros son el texto del mensaje y su longitud. Las sentencias DEFINE que se deben utilizar en el primer parámetro son:

```
#define DB2SEC_LOG_NONE       0   -  No hay registro de anotaciones
#define DB2SEC_LOG_CRITICAL   1   -  Encontrado error grave
#define DB2SEC_LOG_ERROR      2   -  Encontrado error
#define DB2SEC_LOG_WARNING    3   -  Aviso
#define DB2SEC_LOG_INFO       4   -  Informativo
```

Si utiliza la sentencia "define DB2SEC_LOG_INFO", el texto del mensaje solo aparecerá en el archivo `db2diag.log` si el parámetro de configuración *diaglevel* del gestor de bases de datos tiene el valor 4.

Salida:

*void *group_fns*
Es un puntero que apunta a memoria proporcionada por DB2 para una estructura `db2secGroupFunction_1`. En versiones futuras de DB2, pueden existir versiones diferentes de las API, por lo que esto se debe convertir a un puntero que apunte a la estructura `db2secGroupFunction_1` correspondiente a la versión de la API que el plug-in ha implementado.

*char **errormsg*
Es un puntero que indica la dirección de una cadena de caracteres ASCII asignada por el plug-in y que puede ser devuelta en este parámetro si la API no es efectiva.

*db2int32 *errormsglen*
Es puntero que apunta a un valor entero el cual indica la longitud de la cadena de mensaje de error contenida en *char **errormsg*.

Conceptos relacionados:

- "Plug-ins de seguridad" en la página 569
- "Las API de plug-in de seguridad" en la página 597

Información relacionada:

- "Las API de plug-ins de recuperación de grupos" en la página 599

db2secPluginTerm - Depurar recursos de plug-in de grupo

DB2 invoca esta función justo antes de descargar el plug-in. La función debe realizar una depuración apropiada de los recursos que estén retenidos por la biblioteca del plug-in, tal como liberar la memoria asignada por el plug-in, cerrar los archivos que estén todavía abiertos y cerrar las conexiones de red. El plug-in realiza un seguimiento de estos recursos a fin de liberarlos.

Sintaxis de la API de C:

```
SQL_API_RC SQL_API_FN db2secPluginTerm(char **errormsg,
          db2int32  *errormsglen);
```

Salida:

*char **errormsg*
: Es un puntero que indica la dirección de una cadena de caracteres ASCII asignada por el plug-in y que puede ser devuelta en este parámetro si la API no es efectiva.

*db2int32 *errormsglen*
: Es puntero que apunta a un valor entero el cual indica la longitud de la cadena de mensaje de error contenida en *char **errormsg*.

Conceptos relacionados:

- "Plug-ins de seguridad" en la página 569
- "Las API de plug-in de seguridad" en la página 597

Información relacionada:

- "diaglevel - Diagnostic error capture level configuration parameter" en la publicación *Administration Guide: Performance*
- "Las API de plug-ins de recuperación de grupos" en la página 599

db2secGetGroupsForUser - Obtener la lista de grupos del usuario

DB2 invoca esta función para obtener la lista de los grupos a los que pertenece el usuario.

Sintaxis de la API de C:

```
SQL_API_RC SQL_API_FN db2secGetGroupsForUser(
  const char *authid,
  db2int32  authidlen,
  const char *userid,
  db2int32  useridlen,
  const char *usernamespace,
  db2int32  usernamespacelen,
  db2int32  usernamespacetype,
  const char *dbname,
  db2inst32 dbnamelen,
```

```
  const void *token,
  db2int32 tokentype,
  db2int32 location,
  const char *authpluginname,
  db2int32  authpluginnamelen,
  void **grouplist,
  db2int32  *numgroups,
  char **errormsg,
  db2int32  *errormsglen);
```

Entrada:

*const char *authid*
Es el único campo de entrada proporcionado por DB2. El valor de este campo es un ID de autorización de SQL, por lo que su formato adopta la forma de una cadena de caracteres en mayúsculas, sin blancos de cola. El plug-in debe poder devolver una lista de los grupos a los que pertenece el ID de autorización, sin depender de los demás parámetros de entrada. Está permitido devolver una lista abreviada o vacía si esa información no se puede determinar.

Si un usuario no existe, la función debe devolver `DB2SEC_PLUGIN_BADUSER`. DB2 no trata como error el caso de un usuario no existente, pues es admisible que un ID de autorización no tenga ningún grupo asociado a él. Por ejemplo, cuando la función `db2secGetAuthids` devuelve un ID de autorización que no existe en el sistema operativo. El ID de autorización no está asociado a ningún grupo, sin embargo, se le puede seguir asignando privilegios directamente.

Si el plug-in no puede devolver una lista completa de grupos solo a partir del ID de autorización, significa que existen algunas restricciones para determinadas funciones de SQL referentes al soporte de grupos. Consulte la nota contenida en el presente tema, titulada ″Problemas que se pueden producir si la API devuelve una lista de grupos incompleta″, para obtener una lista de los posibles casos de problemas.

db2int32 authidlen
Es la longitud del ID de autorización.

*const char *userid*
Es el ID de usuario correspondiente al ID de autorización. Cuando esta API se invoca en el servidor en una situación de falta de conexión, este campo no se rellena.

db2int32 useridlen
Es la longitud del ID de usuario.

*const void *token*
Es un puntero que apunta a datos proporcionados por el plug-in de autenticación. No es visible para DB2. Permite al creador del plug-in coordinar la información sobre usuario y grupos, si se desea. Esta información no siempre se proporciona en todas las situaciones, en cuyo caso el valor será NULL. Si el plug-in de autenticación utilizado está basado en GSS-API, el símbolo será igual al descriptor de contexto de GSS-API (gss_ctx_id_t).

db2int32 tokentype
Indica el tipo de los datos proporcionados por el plug-in de autenticación. Si el plug-in de autenticación utilizado está basado en GSS-API, el símbolo será igual al descriptor de contexto de GSS-API (gss_ctx_id_t). Si el plug-in

de autenticación está basado en el ID de usuario/contraseña, el tipo de datos será genérico. Vea las siguientes sentencias DEFINE contenidas en `db2secPlugin.h`:

- #define DB2SEC_GENERIC 0 -- Indica que el símbolo procede de un plug-in basado en un ID de usuario/contraseña.
- #define DB2SEC_GSSAPI_CTX_HANDLE 1 -- Indica que el símbolo procede de un plug-in basado en GSS-API (incluido Kerberos).

db2int32 location
Indica si DB2 debe invocar el plug-in del extremo cliente o del extremo servidor. Vea la siguiente sentencia DEFINE contenida en `db2secPlugin.h`:

- #define DB2SEC_SERVER_SIDE 0 -- el plug-in de grupo se invoca en el servidor de bases de datos.
- #define DB2SEC_CLIENT_SIDE 1 -- el plug-in de grupo se invoca en un cliente.

*const char *usernamespace*
Es el espacio de nombres del cual se obtuvo el ID de usuario. Cuando no se conoce el ID de usuario, este campo no se rellena.

db2int32 usernamespacelen
Es la longitud del campo del espacio de nombres.

db2int32 usernamespacetype
Es el tipo del espacio de nombres. Los valores posibles son: DB2SEC_NAMESPACE_SAM_COMPATIBLE (corresponde a un estilo de nombre de usuario tal como torolab\miNombre") o DB2SEC_NAMESPACE_USER_PRINCIPAL (corresponde a un estilo de nombre de usuario tal como miNombre@torolab.ibm.com). Actualmente DB2 solo da soporte a DB2SEC_NAMESPACE_SAM_COMPATIBLE. Cuando no se conoce el ID de usuario, este campo se llena con DB2SEC_USER_NAMESPACE_UNDEFINED. Todas las sentencias DEFINE están situadas en `db2secPlugin.h`.

*const char *dbname*
Es el nombre de la base de datos con la cual se establece conexión.

db2int32 dbnamelen
Es la longitud del nombre de base de datos especificado por *dbname*.

*const char *authpluginname*
Es el nombre del plug-in de autenticación que proporcionó los datos contenidos en el símbolo. El plug-in puede utilizar esta información para determinar las pertenencias correctas a grupos. Esta información no se puede proporcionar si el ID de autorización no está autenticado (por ejemplo, si el ID de autorización no coincide con el usuario conectado actual).

db2int32 authpluginnamelen
Es la longitud del nombre del plug-in de autenticación.

Salida:

*void **grouplist*
La lista de grupos se debe devolver en forma de puntero que apunta a una sección de memoria asignada por el plug-in y que contiene `varchars` concatenados (un `varchar` es una matriz de caracteres en la que el primer byte indica el número de bytes que le siguen a continuación). La longitud es un valor char sin signo y que limita la longitud máxima de un nombre

de grupo a 255 caracteres. Es decir, debido a que estamos utilizando un valor char sin signo (1 byte) para indicar la longitud del nombre de grupo, la longitud máxima es 255. Por ejemplo:

```
"\006GROUP1\007MYGROUP\008MYGROUP3"
```

Cada nombre de grupo debe ser un ID de autorización válido de DB2. La memoria para esta matriz debe ser asignada por el plug-in. Por tanto, el plug-in debe proporcionar una función, tal como la función de plug-in `db2secFreeGroupListMemory()`, que DB2 invocará para liberar la memoria.

*db2int32 *numgroups*
Es el número de grupos contenidos en la lista de grupos.

*char **errormsg*
Es un puntero que indica la dirección de una cadena de caracteres ASCII asignada por el plug-in y que puede ser devuelta en este parámetro si la API no es efectiva.

*db2int32 *errormsglen*
Es puntero que apunta a un valor entero el cual indica la longitud de la cadena de mensaje de error contenida en *char **errormsg*.

Problemas que se pueden producir si la API devuelve una lista de grupos incompleta:

Se pueden producir los problemas siguientes si la API devuelve una lista de grupos incompleta a DB2 UDB:

- Aplicación de SQL incorporado con DYNAMICRULES BIND (o DEFINEDBIND o INVOKEDBIND si los paquetes se están ejecutando como aplicación autónoma). DB2 comprueba la pertenencia a SYSADM y la aplicación falla si depende de la autorización DBADM implícita otorgada por el hecho de ser miembro del grupo SYSADM.
- Se proporciona una autorización alternativa en la sentencia CREATE SCHEMA. La búsqueda de grupos se efectuará respecto al parámetro AUTHORIZATION NAME si existen sentencias CREATE anidadas en la sentencia CREATE SCHEMA.
- Aplicaciones de SQL incorporado con DYNAMICRULES DEFINERUN/DEFINEBIND y los paquetes se están ejecutando en un contexto de rutina. DB2 comprueba la pertenencia a SYSADM del definidor de la rutina y la aplicación falla si depende de la autorización DBADM implícita otorgada por el hecho de ser miembro del grupo SYSADM.
- Proceso de un archivo jar en un entorno MPP. En un entorno MPP, la petición de proceso jar se envía desde el nodo coordinador con el ID de autorización de sesión. El nodo de catálogo recibe las peticiones y procesa los archivos jar basándose en el privilegio otorgado al ID de autorización de la sesión (el usuario que está ejecutando las peticiones de proceso jar).
  - Instalar archivo jar. El ID de autorización de la sesión necesita tener uno de los derechos siguientes: SYSADM, DBADM o CREATEIN (implícito o explícito sobre el esquema jar). La operación falla si los derechos anteriores se otorgan al grupo donde está contenido el ID de autorización de la sesión, pero no se otorgan explícitamente al ID de autorización de la sesión o si solo se posee SYSADM, pues la pertenencia a SYSADM está determinada por la pertenencia al grupo definido por un parámetro de configuración de la base de datos.
  - Eliminar archivo jar. El ID de autorización de la sesión necesita tener uno de los derechos siguientes: SYSADM, DBADM o DROPIN (implícito o explícito

sobre el esquema jar), o bien es el definidor del archivo jar. La operación falla si los derechos anteriores se otorgan al grupo donde está contenido el ID de autorización de la sesión, pero no se otorgan explícitamente al ID de autorización de la sesión, y si el ID de autorización de la sesión no es el definidor del archivo jar o si solo se posee SYSADM, pues la pertenencia a SYSADM está determinada por la pertenencia al grupo definido por un parámetro de configuración de la base de datos.

  - Sustituir el archivo jar. Esto es lo mismo que eliminar el archivo jar, seguido de instalar el archivo jar. Son aplicables las dos situaciones anteriores.

- Regenerar vistas. Esto es desencadenado por las sentencias ALTER TABLE, ALTER COLUMN, SET DATA TYPE VARCHAR/VARGRAPHIC, o durante la migración. DB2 comprueba la pertenencia a SYSADM del definidor de la vista. La aplicación falla si depende de la autorización DBADM implícita otorgada por el hecho de ser miembro del grupo SYSADM.
- Cuando se emite la sentencia SET SESSION_USER. Las operaciones subsiguientes de DB2 se ejecutan bajo el contexto del ID de autorización especificado por esta sentencia. Estas operaciones fallan si los privilegios necesarios que son propiedad de uno de los miembros del grupo SESSION_USER no se otorga explícitamente al ID de autorización SESSION_USER.

Conceptos relacionados:

- "Plug-ins de seguridad" en la página 569
- "Las API de plug-in de seguridad" en la página 597

Información relacionada:

- "Las API de plug-ins de recuperación de grupos" en la página 599

db2secDoesGroupExist - Comprobar si existe el grupo

Esta función se utiliza para determinar si un ID de autorización representa un grupo. Si el nombre de grupo existe, la función devuelve DB2SEC_PLUGIN_OK, para indicar un resultado satisfactorio. Si el nombre de grupo no es válido, la función devuelve DB2SEC_PLUGIN_INVALIDUSERORGROUP. La API también puede devolver DB2SEC_PLUGIN_GROUPSTATUSNOTKNOWN cuando no se puede determinar si los datos de entrada constituyen un grupo válido. Si se devuelve un grupo no válido o desconocido, DB2 no puede determinar si el ID de autorización es un grupo o usuario cuando se emite la sentencia GRANT sin las palabras clave USER y GROUP, por lo que el usuario recibe el error SQLCODE -569, SQLSTATE 56092.

Sintaxis de la API de C:

```
SQL_API_RC SQL_API_FN db2secDoesGroupExist(
 const char *groupname,
 db2int32  groupnamelen,
 char **errormsg,
 db2int32  *errormsglen
);
```

Entrada:

*const char *groupname*
Es un ID de autorización, escrito en mayúsculas y sin blancos de cola.

db2int32 groupnamelen
Es la longitud del nombre de grupo.

Salida:

*char **errormsg*
Es un puntero que indica la dirección de una cadena de caracteres ASCII asignada por el plug-in y que puede ser devuelta en este parámetro si la API no es efectiva.

*db2int32 *errormsglen*
Es puntero que apunta a un valor entero el cual indica la longitud de la cadena de mensaje de error contenida en *char **errormsg*.

Conceptos relacionados:

- "Plug-ins de seguridad" en la página 569
- "Las API de plug-in de seguridad" en la página 597

Información relacionada:

- "Las API de plug-ins de recuperación de grupos" en la página 599

db2secFreeGroupListMemory - Liberar memoria de lista de grupo

Esta función indica a la biblioteca del plug-in que la memoria a la cual apunta `ptr` ya no es necesaria para DB2. El plug-in necesita liberar esta memoria.

Sintaxis de la API de C:

```
SQL_API_RC SQL_API_FN db2secFreeGroupListMemory(
 void *ptr
 char **errormsg,
 db2int32  *errormsglen);
```

Entrada:

*void *ptr*
Es un puntero que apunta a la memoria que se debe liberar.

Salida:

*char **errormsg*
Es un puntero que indica la dirección de una cadena de caracteres ASCII asignada por el plug-in y que puede ser devuelta en este parámetro si la API no es efectiva.

*db2int32 *errormsglen*
Es puntero que apunta a un valor entero el cual indica la longitud de la cadena de mensaje de error contenida en *char **errormsg*.

Conceptos relacionados:

- "Plug-ins de seguridad" en la página 569
- "Las API de plug-in de seguridad" en la página 597

Información relacionada:

- "Las API de plug-ins de recuperación de grupos" en la página 599

db2secFreeErrormsg - Liberar la memoria de mensajes de error

Esta función es invocada por DB2 para liberar la memoria donde se almacenan los mensajes de error procedentes de una llamada anterior a una API de plug-in. Esta es la única API que no devuelve también un mensaje de error. Si esta función devuelve un error, DB2 lo registrará y continuará su proceso.

Sintaxis de la API de C:

```
SQL_API_RC SQL_API_FN db2secFreeErrormsg(char *msgtobefree);
```

Entrada:

*char *msgtobefree*
Es un puntero que apunta al mensaje de error procedente de una llamada anterior a la API.

Conceptos relacionados:

- "Plug-ins de seguridad" en la página 569
- "Las API de plug-in de seguridad" en la página 597

Información relacionada:

- "Las API de plug-ins de recuperación de grupos" en la página 599

Las API de plug-in de autenticación de usuario

Las API del plug-in de autenticación por ID usuario/contraseña

Para la biblioteca de plug-in de autenticación por ID de usuario/contraseña, debe implementar las API siguientes en el extremo del cliente:

```
SQL_API_RC SQL_API_FN db2secClientAuthPluginInit(
  db2int32 version,
  void *client_fns,
  db2secLogMessage *logMessage_fn,
  char **errormsg,
  db2int32  *errormsglen);
```

Nota: La función mostrada más arriba utiliza como entrada un puntero que apunta a una función, `*logMessage_fn`, con el prototipo siguiente:

```
SQL_API_RC (SQL_API_FN db2secLogMessage) (
  db2int32 level,
  void *data,
  db2int32 length);
```

```
SQL_API_RC SQL_API_FN db2secClientAuthPluginTerm(
  char **errormsg,
  db2int32  *errormsglen);

/* Solo utilizado para gssapi: */
db2secGenerateInitialCred(
  const char *userid,
  db2int32  useridlen,
  const char *usernamespace,
  db2int32  usernamespacelen,
```

```
|                           db2int32  usernamespacetype,
|                           const char *password,
|                           db2int32  passwordlen,
|                           const char *newpassword,
|                           db2int32  newpasswordlen,
|                           const char *dbname,
|                           db2int32  dbnamelen,
|                           gss_cred_id_t *pGSSCredHandle,
|                           void **initInfo,
|                           char **errormsg,
|                           db2int32  *errormsglen);
|
|                         /* Opcional */
|                         SQL_API_RC SQL_API_FN db2secRemapUserid(
|                           char userid[DB2SEC_MAX_USERID_LENGTH],
|                           db2int32  *useridlen,
|                           db2int32  useridtype,
|                           char usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH],
|                           db2int32  *usernamespacelen,
|                           db2int32  *usernamespacetype,
|                           char password[DB2SEC_MAX_PASSWORD_LENGTH],
|                           db2int32  *passwordlen,
|                           char newpassword[DB2SEC_MAX_PASSWORD_LENGTH],
|                           db2int32  *newpasswordlen,
|                           const char *dbname,
|                           db2int32  dbnamelen,
|                           char **errormsg,
|                           db2int32  *errormsglen);
|                         SQL_API_RC SQL_API_FN db2secGetDefaultLoginContext(
|                           char authid[DB2SEC_MAX_AUTHID_LENGTH],
|                           db2int32  *authidlen,
|                           char userid[DB2SEC_MAX_USERID_LENGTH],
|                           db2int32  *useridlen,
|                           db2int32 useridtype,
|                           char usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH],
|                           db2int32  *usernamespacelen,
|                           db2int32  *usernamespacetype,
|                           const char *dbname,
|                           db2int32  dbnamelen,
|                           void **token,
|                           char **errormsg,
|                           db2int32  *errormsglen);
|                         SQL_API_RC SQL_API_FN db2secValidatePassword(
|                           const char *userid,
|                           db2int32  useridlen,
|                           const char *usernamespace,
|                           db2int32  usernamespacelen,
|                           db2int32  usernamespacetype,
|                           const char *password,
|                           db2int32  passwordlen,
|                           const char *newpassword,
|                           db2int32  newpasswordlen,
|                           const char *dbname,
|                           db2int32  dbnamelen,
|                           db2Uint32   connection_details,
|                           void **token,
|                           char **errormsg,
|                           db2int32  *errormsglen);
|                         /* Solo para GSS-API */
|                         SQL_API_RC SQL_API_FN db2secProcessServerPrincipalName(
|                           const void *name,
|                           db2int32 nameLen,
|                           gss_name_t *gssName,
|                           char **errormsg,
|                           db2int32  *errormsglen);
```

```
/* Funciones para liberar memoria retenida por la DLL */
SQL_API_RC SQL_API_FN db2secFreeToken(
  void *token,
  char **errormsg,
  db2int32  *errormsglen);

SQL_API_RC SQL_API_FN db2secFreeErrormsg(char *errormsg);

SQL_API_RC SQL_API_FN db2secFreeInitInfo(
  void *initInfo,
  char **errormsg,
  db2int32  *errormsglen);
```

La única API que se debe poder resolver externamente es `db2secClientAuthPluginInit()`. Esta función utiliza `void *` como parámetro de entrada, que se debe convertir a uno de estos dos tipos:

```
typedef struct db2secUseridPasswordClientAuthFunctions_1
{
   db2int32 version;
   db2int32 plugintype;

SQL_API_RC (SQL_API_FN * db2secGetDefaultLoginContext)(
  char authid[DB2SEC_MAX_AUTHID_LENGTH],
  db2int32  *authidlen,
  char userid[DB2SEC_MAX_USERID_LENGTH],
  db2int32  *useridlen,
  db2int32 useridtype,
  char usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH],
  db2int32  *usernamespacelen,
  db2int32  *usernamespacetype,
  const char *dbname,
  db2int32  dbnamelen,
  void **token,
  char **errormsg,
  db2int32  *errormsglen);

SQL_API_RC (SQL_API_FN * db2secRemapUserid)(  // Optional
  char userid[DB2SEC_MAX_USERID_LENGTH],
  db2int32  *useridlen,
  char usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH],
  db2int32  *usernamespacelen,
  db2int32  *usernamespacetype,
  char password[DB2SEC_MAX_PASSWORD_LENGTH],
  db2int32  *passwordlen,
  char newpassword[DB2SEC_MAX_PASSWORD_LENGTH],
  db2int32  *newpasswordlen,
  const char *dbname,
  db2int32  dbnamelen,
  char **errormsg,
  db2int32  *errormsglen);

SQL_API_RC (SQL_API_FN * db2secValidatePassword)(
  const char *userid,
  db2int32  useridlen,
  const char *usernamespace,
  db2int32  usernamespacelen,
  db2int32  usernamespacetype,
  const char *password,
  db2int32  passwordlen,
  const char *newpassword,
  db2int32  newpasswordlen,
  const char *dbname,
  db2int32  dbnamelen,
  db2Uint32   connection_details,
  void **token,
  char **errormsg,
  db2int32  *errormsglen);
```

```
SQL_API_RC (SQL_API_FN * db2secFreeToken)(
  void **token,
  char **errormsg,
  db2int32  *errormsglen);

SQL_API_RC (SQL_API_FN * db2secFreeErrormsg)(char *errormsg);


SQL_API_RC (SQL_API_FN * db2secClientAuthPluginTerm)(
  char **errormsg,
  db2int32  *errormsglen);
}
```

o bien

```
typedef struct db2secGssapiClientAuthFunctions_1
{
  db2int32 version;
  db2int32 plugintype;

SQL_API_RC (SQL_API_FN * db2secGetDefaultLoginContext) (
  char authid[DB2SEC_MAX_AUTHID_LENGTH],
  db2int32  *authidlen,
  char userid[DB2SEC_MAX_USERID_LENGTH],
  db2int32  *useridlen,
  db2int32 useridtype,
  char usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH],
  db2int32  *usernamespacelen,
  db2int32  *usernamespacetype,
  const char *dbname,
  db2int32  dbnamelen,
  void **token,
  char **errormsg,
  db2int32  *errormsglen);

SQL_API_RC (SQL_API_FN * db2secProcessServerPrincipalName) (
  const void *data,
  gss_name_t *gssName,
  char **errormsg,
  db2int32  *errormsglen);

SQL_API_RC (SQL_API_FN * db2secGenerateInitialCred) (
  const char *userid,
  db2int32 useridlen,
  const char *usernamespace,
  db2int32 usernamespacelen,
  db2int32 usernamespacetype,
  const char *password,
  db2int32 passwordlen,
  const char *newpassword,
  db2int32 newpasswordlen,
  const char *dbname,
  db2int32 dbnamelen,
  gss_cred_id_t *pGSSCredHandle,
  void **initInfo,
  char **errormsg,
  db2int32 *errormsglen);

SQL_API_RC (SQL_API_FN * db2secFreeToken)(
  void *token,
  char **errormsg,
  db2int32  *errormsglen);

SQL_API_RC (SQL_API_FN * db2secFreeErrormsg)(char *errormsg);
```

```
SQL_API_RC (SQL_API_FN * db2secFreeInitInfo) (
  void *initInfo,
  char **errormsg,
  db2int32  *errormsglen);

SQL_API_RC (SQL_API_FN * db2secClientAuthPluginTerm) (
  char **errormsg,
  db2int32  *errormsglen);

/* Funciones específicas de GSS-API  -- consulte
  db2secPlugin.h  para obtener la lista de parámetros */

  OM_uint32 (SQL_API_FN * gss_init_sec_context )(<lista de parámetros>);
  OM_uint32 (SQL_API_FN * gss_delete_sec_context )(<lista de parámetros>);
  OM_uint32 (SQL_API_FN * gss_display_status )(<lista de parámetros>);
  OM_uint32 (SQL_API_FN * gss_release_buffer )(<lista de parámetros>);
  OM_uint32 (SQL_API_FN * gss_release_cred )(<lista de parámetros>);
  OM_uint32 (SQL_API_FN * gss_release_name )(<lista de parámetros>);
  }
```

Debe utilizar `db2secUseridPasswordClientAuthFunctions_1` si está escribiendo un plug-in de ID de usuario/contraseña. Debe utilizar `db2secGssapiClientAuthFunctions_1` si está escribiendo un plug-in de GSS_API (incluido Kerberos).

Para la biblioteca de plug-in de autenticación por ID de usuario/contraseña, debe implementar las API siguientes en el extremo del servidor:

```
db2secServerAuthPluginInit(
  db2int32 version,
  void *server_fns,
  db2secGetConDetails *getConDetails_fn,
  db2secLogMessage *logMessage_fn,
  char **errormsg,
  db2int32  *errormsglen);
```

La función anterior utiliza como entrada un puntero que apunta a una función `*logMessage_fn` y a una función `*getConDetails_fn`, con los prototipos siguientes:

```
SQL_API_RC (SQL_API_FN db2secLogMessage) (
  db2int32 level,
  void *data,
  db2int32 length);

SQL_API_RC (SQL_API_FN db2secGetConDetails)(
  db2int32 conDetailsVersion,
  const void *pConDetails);
```

Luego, esta función utiliza como segundo parámetro de entrada el puntero `pConDetails`, que apunta a una estructura definida de esta manera:

```
typedef struct db2sec_con_details_1
{
  db2int32 clientProtocol;
  db2Uint32 clientIPAddress;
  db2Uint32 connect_info_bitmap;
  db2int32 dbnameLen;
  char dbname[DB2SEC_MAX_DBNAME_LENGTH + 1];
} db2sec_con_details_1;
```

Consulte la sección sobre la descripción detallada para obtener una explicación de esta función y estructura.

```
db2secServerAuthPluginTerm(
  char **errormsg,
  db2int32  *errormsglen);
```

```
SQL_API_RC SQL_API_FN db2secValidatePassword(
  const char *userid,
  db2int32  useridlen,
  const char *usernamespace,
  db2int32  usernamespacelen,
  db2int32  usernamespacetype,
  const char *password,
  db2int32  passwordlen,
  const char *newpasswd,
  db2int32  newpasswdlen,
  const char *dbname,
  db2int32  dbnamelen,
  db2Uint32  connection_details,
  void **token,
  char **errormsg,
  db2int32  *errormsglen);
SQL_API_RC SQL_API_FN db2secGetAuthIDs(
  const char *userid,
  db2int32  useridlen,
  const char *usernamespace,
  db2int32  usernamespacelen,
  db2int32  usernamespacetype,
  const char *dbname,
  db2int32  dbnamelen,
  void **token,
  char SystemAuthid[DB2SEC_MAX_AUTHID_LENGTH],
  db2int32  SystemAuthidlen,
  char InitialSessionAuthID[DB2SEC_MAX_AUTHID_LENGTH],
  db2int32  *InitialSessionAuthIdlen,
  char username[DB2SEC_MAX_USERID_LENGTH],
  db2int32  *usernamelen,
  db2int32 *initsessionidtype,
  char **errormsg,
  db2int32  *errormsglen);
SQL_API_RC SQL_API_FN db2secDoesAuthIDExist(
  const char *authid,
  db2int32 authidlen,
  const char *errormsg,
  db2int32  *errormsglen);
SQL_API_RC SQL_API_FN db2secFreeToken(
  void *token,
  char **errormsg,
  db2int32  *errormsglen);
SQL_API_RC SQL_API_FN db2secFreeErrormsg(char *errormsg);
```

La única API que se debe poder resolver externamente es `db2secServerAuthPluginInit()`. Esta función utiliza `void *` como parámetro de entrada, que se debe convertir a uno de estos dos tipos:

```
typedef struct db2secUseridPasswordServerAuthFunctions_1
{
  db2int32 version;
  db2int32 plugintype;

  /* Por razones de legibilidad, las listas de parámetros se han
     dejado en blanco; vea más arriba para conocer los parámetros */
 SQL_API_RC (SQL_API_FN * db2secValidatePassword)(<lista de parámetros>);
  SQL_API_RC (SQL_API_FN * db2secGetAuthIDs)(<lista de parámetros);
  SQL_API_RC (SQL_API_FN * db2secDoesAuthIDExist)(<lista de parámetros>);
  SQL_API_RC (SQL_API_FN * db2secFreeToken)(<lista de parámetros>);
  SQL_API_RC (SQL_API_FN * db2secFreeErrormsg)(<lista de parámetros>);
 SQL_API_RC (SQL_API_FN * db2secServerAuthPluginTerm)();
} userid_password_server_auth_functions;
```

o bien

```
typedef struct db2secGssapiServerAuthFunctions_1
{
 db2int32 version;
  db2int32 plugintype;
  gss_buffer_desc serverPrincipalName;
  gss_cred_id_t ServerCredHandle;
  SQL_API_RC (SQL_API_FN * db2secGetAuthIDs)(<lista de parámetros);
  SQL_API_RC (SQL_API_FN * db2secDoesAuthIDExist)(<lista de parámetros>);
  SQL_API_RC (SQL_API_FN * db2secFreeToken)(<lista de parámetros>);
  SQL_API_RC (SQL_API_FN * db2secFreeErrormsg)(<lista de parámetros>);
  SQL_API_RC (SQL_API_FN * db2secServerAuthPluginTerm)();

 /* Funciones específicas de GSS-API;
   consulte db2secPlugin.h para obtener la lista de parámetros*/
 OM_uint32 (SQL_API_FN * gss_accept_sec_context )(<lista de parámetros>);
 OM_uint32 (SQL_API_FN * gss_display_name )(<lista de parámetros>);
 OM_uint32 (SQL_API_FN * gss_delete_sec_context )(<lista de parámetros>);
 OM_uint32 (SQL_API_FN * gss_display_status )(<lista de parámetros>);
 OM_uint32 (SQL_API_FN * gss_release_buffer )(<lista de parámetros>);
 OM_uint32 (SQL_API_FN * gss_release_cred )(<lista de parámetros>);
 OM_uint32 (SQL_API_FN * gss_release_name )(<lista de parámetros>);

} gssapi_server_auth_functions;
```

Debe utilizar db2secUseridPasswordServerAuthFunctions_1 si está escribiendo un plug-in de ID de usuario/contraseña. Debe utilizar db2secGssapiServerAuthFunctions_1 si está escribiendo un plug-in de GSS_API (incluido Kerberos).

Conceptos relacionados:

- "Plug-ins de seguridad" en la página 569
- "Las API de plug-in de seguridad" en la página 597

Tareas relacionadas:

- "Despliegue de un plug-in de ID de usuario/contraseña" en la página 581

Información relacionada:

- "db2secGetGroupsForUser - Obtener la lista de grupos del usuario" en la página 602
- "db2secClientAuthPluginInit - Inicializar el plug-in de autenticación del cliente" en la página 615
- "db2secClientAuthPluginTerm - Depurar recursos de plug-in de autenticación de cliente" en la página 616
- "db2secRemapUserid - Reasignar ID de usuario y contraseña" en la página 617
- "db2secGetDefaultLoginContext - Obtener contexto de conexión por omisión" en la página 618
- "db2secGenerateInitialCred - Generar credenciales iniciales" en la página 620
- "db2secValidatePassword - Validar contraseña" en la página 622
- "db2secProcessServerPrincipalName - Procesar nombre de principal de servicio devuelto desde servidor" en la página 624
- "db2secFreeToken - Liberar memoria retenida por símbolo (token)" en la página 625
- "db2secFreeInitInfo - Liberar recursos retenidos por db2secGenerateInitialCred()" en la página 626
- "db2secServerAuthPluginInit - Inicializar el plug-in de autenticación del servidor" en la página 627

- "db2secServerAuthPluginTerm - Depurar recursos de plug-in de autenticación de servidor" en la página 629
- "db2secGetAuthIDs - Obtener los ID de autenticación" en la página 629
- "db2secDoesAuthIDExist - Comprobar si existe el ID de autenticación" en la página 631

db2secClientAuthPluginInit - Inicializar el plug-in de autenticación del cliente

Esta es la función de inicialización correspondiente a la biblioteca de plug-in que DB2 invoca inmediatamente después de cargar la biblioteca. El puntero `functions` se debe convertir a la estructura `client_auth_functions` apropiada para la versión de interfaz.

Sintaxis de la API de C:

```
SQL_API_RC SQL_API_FN db2secClientAuthPluginInit(
 db2int32 version,
 void *client_fns,
 db2secLogMessage *logMessage_fn,
 char **errormsg,
 db2int32  *errormsglen);
```

Entrada:

db2int32 version
Es el número de versión más alto de la API al que DB2 da soporte actualmente.

*db2secLogMessage *logMessage_fn*
Es un puntero que apunta a una función proporcionada por DB2. El plug-in puede invocar esta función para registrar en `db2diag.log` más información sobre errores para utilizarla con fines de depuración o información. El primer parámetro debe utilizar la sentencia DEFINE contenida en `db2secPlugin.h` y los dos últimos parámetros son el texto del mensaje y su longitud. Las sentencias DEFINE que se deben utilizar en el primer parámetro son:

```
#define DB2SEC_LOG_NONE       0   -  No hay registro de anotaciones
#define DB2SEC_LOG_CRITICAL   1   -  Encontrado error grave
#define DB2SEC_LOG_ERROR      2   -  Encontrado error
#define DB2SEC_LOG_WARNING    3   -  Aviso
#define DB2SEC_LOG_INFO       4   -  Informativo
```

Si utiliza la sentencia "define DB2SEC_LOG_INFO", el texto del mensaje solo aparecerá en el archivo `db2diag.log` si el parámetro de configuración *diaglevel* del gestor de bases de datos tiene el valor 4.

Salida:

*void *client_fns*
Es un puntero que apunta a memoria proporcionada por DB2 para una estructura `client_auth_functions`. En versiones futuras de DB2, pueden existir versiones diferentes de las API, por lo que esto se debe convertir a un puntero que apunte a la estructura `client_auth_functions` correspondiente a la versión de la API que el plug-in implementa.

*char **errormsg*
Es un puntero que indica la dirección de una cadena de caracteres ASCII asignada por el plug-in y que puede ser devuelta en este parámetro si la API no es efectiva.

*db2int32 *errormsglen*
Es puntero que apunta a un valor entero el cual indica la longitud de la cadena de mensaje de error contenida en *char **errormsg*.

Conceptos relacionados:

- "Plug-ins de seguridad" en la página 569
- "Las API de plug-in de seguridad" en la página 597

Información relacionada:

- "diaglevel - Diagnostic error capture level configuration parameter" en la publicación *Administration Guide: Performance*
- "Las API del plug-in de autenticación por ID usuario/contraseña" en la página 608

db2secClientAuthPluginTerm - Depurar recursos de plug-in de autenticación de cliente

DB2 invoca esta función justo antes de descargar el plug-in. La función debe realizar una depuración apropiada de los recursos que estén retenidos por la biblioteca del plug-in, tal como liberar la memoria asignada por el plug-in, cerrar los archivos que estén todavía abiertos y cerrar las conexiones de red. El plug-in realiza un seguimiento de estos recursos a fin de liberarlos.

Sintaxis de la API de C:

```
SQL_API_RC SQL_API_FN db2secClientAuthPluginTerm(
 char **errormsg
 db2int32  *errormsglen);
```

Salida:

*char **errormsg*
Es un puntero que indica la dirección de una cadena de caracteres ASCII asignada por el plug-in y que puede ser devuelta en este parámetro si la API no es efectiva.

*db2int32 *errormsglen*
Es puntero que apunta a un valor entero el cual indica la longitud de la cadena de mensaje de error contenida en *char **errormsg*.

Conceptos relacionados:

- "Plug-ins de seguridad" en la página 569
- "Las API de plug-in de seguridad" en la página 597

Información relacionada:

- "Las API del plug-in de autenticación por ID usuario/contraseña" en la página 608

db2secRemapUserid - Reasignar ID de usuario y contraseña

Esta función se invoca en el extremo cliente para poder reasignar un ID de usuario y contraseña dados (y posiblemente una nueva contraseña y un nuevo espacio de nombres) a valores diferentes de los proporcionados en el momento de la conexión. DB2 solo invoca esta función si durante la conexión se proporciona como mínimo un ID de usuario y una contraseña. Esto impide que un plug-in reasigne por sí mismo un ID de usuario a un par ID usuario/contraseña. Esta función es opcional y no se invoca si no se proporciona.

Sintaxis de la API de C:

```
SQL_API_RC SQL_API_FN db2secRemapUserid(
  char userid[DB2SEC_MAX_USERID_LENGTH],
  db2int32  *useridlen,
  char usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH],
  db2int32  *usernamespacelen,
  db2int32  *usernamespacetype,
  char password[DB2SEC_MAX_PASSWORD_LENGTH],
  db2int32  *passwordlen,
  char newpassword[DB2SEC_MAX_PASSWORD_LENGTH],
  db2int32  *newpasswordlen,
  const char *dbname,
  db2int32  dbnamelen,
  char **errormsg,
  db2int32  *errormsglen);
```

Entrada:

*const char *dbname*
Es el nombre de la base de datos a la cual se conecta el cliente.

db2int32 dbnamelen
Es la longitud del nombre de la base de datos.

Entrada/salida:

char userid[DB2SEC_MAX_USERID_LENGTH]
Es el ID de usuario que se debe reasignar. Si existe un ID de usuario como valor de entrada, debe existir un ID de usuario como valor de salida que puede ser igual o diferente que el ID de usuario de entrada. Si no existe un ID de usuario como valor de entrada, el plug-in no devuelve un ID de usuario como valor de salida.

*db2int32 *useridlen*
Es la longitud del ID de usuario devuelto en el parámetro userid.

char usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH]
Es el espacio de nombres del cual procede el ID de usuario. Es opcional resasignar este valor. Si usernamespace no se proporcionó como valor de entrada de esta función y la función proporciona un valor como salida, el espacio de nombres de usuario (usernamespace) solo será utilizado por DB2 para la autenticación de cliente y no se tendrá en cuenta para otros tipos de autenticación.

*db2int32 *usernamespacelen*
Es la longitud antigua y nueva del espacio de nombres de usuario. De acuerdo con la limitación de que usernamespacetype debe ser DB2SEC_NAMESPACE_SAM_COMPATIBLE, la longitud máxima soportada en la versión actual de DB2 es 15 bytes.

*db2int32 *usernamespacetype*
Es el tipo antiguo y nuevo del espacio de nombres. En la versión actual de DB2, el único tipo soportado de espacio de nombres es DB2SEC_NAMESPACE_SAM_COMPATIBLE.

char password[DB2SEC_MAX_PASSWORD_LENGTH]
Es la contraseña que se debe reasignar. Si se ha proporcionado una contraseña como entrada, debe existir una contraseña de salida, que puede ser diferente. Si no se ha proporcionado una contraseña como entrada, el plug-in no devuelve una contraseña de salida.

*db2int32 *passwordlen*
Es la longitud de la contraseña.

char newpassword[DB2SEC_MAX_PASSWORD_LENGTH]
Es una contraseña nueva, si la contraseña se debe cambiar.

Nota: Esta es la nueva contraseña que DB2 colocará en el campo newpassword de la función db2secValidatePassword en el cliente o servidor (dependiendo del valor del parámetro de configuración AUTHENTICATION del gestor de bases de datos). Si se ha proporcionado una nueva contraseña como entrada, debe existir una nueva contraseña de salida, que puede ser una nueva contraseña diferente. Si no se ha proporcionado una nueva contraseña como entrada, el plug-in no devuelve una nueva contraseña de salida.

*db2int32 *newpasswordlen*
Es la longitud de la nueva contraseña.

Salida:

*char **errormsg*
Es un puntero que indica la dirección de una cadena de caracteres ASCII asignada por el plug-in y que puede ser devuelta en este parámetro si la API no es efectiva.

*db2int32 *errormsglen*
Es puntero que apunta a un valor entero el cual indica la longitud de la cadena de mensaje de error contenida en *char **errormsg*.

Conceptos relacionados:

- "Plug-ins de seguridad" en la página 569
- "Las API de plug-in de seguridad" en la página 597

Información relacionada:

- "Las API del plug-in de autenticación por ID usuario/contraseña" en la página 608

db2secGetDefaultLoginContext - Obtener contexto de conexión por omisión

DB2 invoca esta función para determinar el usuario asociado al contexto de conexión por omisión, es decir, para determinar el ID de autorización de DB2 del usuario que emite un mandato de DB2 sin especificar explícitamente un ID de usuario (autenticación implícita para una base de datos o una autorización local). Esta función devuelve normalmente un ID de autorización y un ID de usuario.

Sintaxis de la API de C:

```
db2secGetDefaultLoginContext(
 char authid[DB2SEC_MAX_AUTHID_LENGTH],
 db2int32  *authidlen,
 char userid[DB2SEC_MAX_USERID_LENGTH],
 db2int32  *useridlen,
 db2int32  useridtype,
 char usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH],
 db2int32  *usernamespacelen,
 db2int32  *usernamespacetype,
 const char *dbname,
 db2int32  dbnamelen,
 void **token,
 char **errormsg,
 db2int32  *errormsglen);
```

Entrada:

*const char *dbname*
Es el nombre de la base de datos con la que se establece conexión si la llamada se utiliza en el contexto de una conexión de base de datos. Para acciones de autorización local o conexiones físicas de instancia, este parámetro tendrá el valor NULL.

db2int32 dbnamelen
Es la longitud del nombre de la base de datos.

db2int32 useridtype
Especifica si DB2 desea el usuario real o efectivo del proceso.

Salida:

char authid[DB2SEC_MAX_AUTHID_LENGTH]
Es el campo en el que se devuelve el ID de autorización. El valor devuelto debe cumplir los convenios de denominación para los ID de autorización de DB2, de lo contrario el usuario no tendrá autorización para ejecutar la acción solicitada.

*db2int32 *authidlen*
Es la longitud del ID de autorización devuelto.

char userid[DB2SEC_MAX_USERID_LENGTH]
Es el campo en el que se devuelve el ID de usuario.

*db2int32 *useridlen*
Es la longitud del ID de usuario devuelto.

*void **token*
Es un puntero que apunta a datos asignados por el plug-in para pasarlos a llamadas de autenticación subsiguientes del plug-in, o posiblemente al plug-in de recuperación de grupos. La estructura de estos datos la debe determinar el creador del plug-in.

char usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH]
Es la longitud del espacio de nombres devuelto. De acuerdo con la limitación de que usernamespacetype debe ser DB2SEC_NAMESPACE_SAM_COMPATIBLE, la longitud máxima soportada en la versión actual de DB2 es 15 bytes.

*db2int32 *usernamespacelen*
Es la longitud del espacio de nombres devuelto. De acuerdo con la limitación de que usernamespacetype debe ser DB2SEC_NAMESPACE_SAM_COMPATIBLE, la longitud máxima soportada en la versión actual de DB2 es 15 bytes.

*db2int32 *usernamespacetype*
Es aplicable lo descrito más arriba. En la versión actual de DB2, el único tipo soportado de espacio de nombres es DB2SEC_NAMESPACE_SAM_COMPATIBLE.

*char **errormsg*
Es un puntero que indica la dirección de una cadena de caracteres ASCII asignada por el plug-in y que puede ser devuelta en este parámetro si la API no es efectiva.

*db2int32 *errormsglen*
Es puntero que apunta a un valor entero el cual indica la longitud de la cadena de mensaje de error contenida en *char **errormsg*.

Conceptos relacionados:

- "Plug-ins de seguridad" en la página 569
- "Las API de plug-in de seguridad" en la página 597

Información relacionada:

- "authentication - Authentication type configuration parameter" en la publicación *Administration Guide: Performance*
- "Las API del plug-in de autenticación por ID usuario/contraseña" en la página 608

db2secGenerateInitialCred - Generar credenciales iniciales

Esta función obtiene las credenciales iniciales de GSS-API basándose en el ID de usuario y contraseña que se han pasado. En el caso de Kerberos, el valor devuelto será TGT. El descriptor de credencial que se devuelve en `pGSSCredHandle` es el descriptor que se utilizará con `gss_init_sec_context()` y debe ser una credencial INITIATE o BOTH. Esta función solo se invoca cuando se proporciona un ID de usuario y posiblemente una contraseña. En otro caso, DB2 especificará `GSS_C_NO_CREDENTIAL` al invocar `gss_init_sec_context()` para indicar que se debe utilizar la credencial por omisión obtenida del contexto de conexión actual.

Sintaxis de la API de C:

```
db2secGenerateInitialCred(
 const char *userid,
 db2int32  useridlen,
 const char *usernamespace,
 db2int32  usernamspacelen,
 db2int32  usernamespacetype,
 const char *password,
 db2int32  passwordlen,
 const char *newpassword,
 db2int32  newpasswordlen,
 const char *dbname,
 db2int32  dbnamelen,
 gss_cred_id_t *pGSSCredHandle,
 void **initInfo,
 char **errormsg,
 db2int32  *errormsglen);
```

Entrada:

*const char *userid*
Es el ID de usuario cuya contraseña se debe verificar.

db2int32 useridlen
Es la longitud del ID de usuario.

*const char *usernamespace*
Es el espacio de nombres del cual se obtuvo el ID de usuario.

db2int32 usernamespacelen
Es la longitud del campo del espacio de nombres.

db2int32 usernamespacetype
Es el tipo del espacio de nombres.

*const char *password*
Es la contraseña que se debe verificar. DB2 cifra esta contraseña antes de pasarla al plug-in.

db2int32 passwordlen
Es la longitud de la nueva contraseña.

*const char *newpassword*
Es una contraseña nueva, si la contraseña se debe cambiar. Si no se solicita cambio de contraseña, el valor será NULL. Si el valor no es NULL, la función debe validar la contraseña antigua antes de cambiar a la nueva contraseña. El plug-in puede no satisfacer una petición de cambio de contraseña, pero si no satisface la petición, debe devolver inmediatamente DB2SEC_PLUGIN_CHANGEPASSWORD_NOTSUPPORTED sin validar la contraseña antigua.

db2int32 newpasswordlen
Es la longitud de la nueva contraseña.

*const char *dbname*
Es el nombre de la base de datos con la cual se establece conexión. La función puede hacer caso omiso de este campo, o puede devolver DB2SEC_PLUGIN_CONNECTION_DISALLOWED si desea limitar el acceso a determinadas bases de datos para usuarios que por lo demás tienen contraseñas validas.

db2int32 dbnamelen
Es la longitud del nombre de la base de datos.

Salida:

*gss_cred_id_t *pGSSCredHandle*
Es un puntero que apunta al descriptor de credencial de GSS-API.

*void **initInfo*
Es un puntero que apunta a datos que son opacos para DB2. El plug-in puede utilizar este puntero para mantener una lista de recursos que se asignan durante el proceso de generar el descriptor de credencial. DB2 invoca `db2secFreeInitInfo()` al final de la autenticación, en cuyo momento el plug-in puede entonces liberar estos recursos. Si el plug-in no necesita mantener una lista como esa, devolverá un valor NULL.

*char **errormsg*
Es un puntero que indica la dirección de una cadena de caracteres ASCII asignada por el plug-in y que puede ser devuelta en este parámetro si la API no es efectiva.

Nota: Para esta API, no se deben crear mensajes de error si el valor de retorno indica la existencia de un ID de usuario o contraseña

incorrectos. Solamente se debe devolver un mensaje de error si existe un error interno en la API que le impida finalizar debidamente su ejecución.

*db2int32 *errormsglen*
Es puntero que apunta a un valor entero el cual indica la longitud de la cadena de mensaje de error contenida en *char **errormsg*.

Conceptos relacionados:

- "Plug-ins de seguridad" en la página 569
- "Las API de plug-in de seguridad" en la página 597

Información relacionada:

- "Las API del plug-in de autenticación por ID usuario/contraseña" en la página 608

db2secValidatePassword - Validar contraseña

Esta función proporciona un método de autenticación basado en un ID de usuario y contraseña durante una operación de conexión a una base de datos.

Nota: El código de plug-in se ejecuta con los privilegios de la aplicación cliente. El creador del plug-in debe tener esto en cuenta si la autenticación necesita privilegios especiales (tales como root).

La API devuelve DB2SEC_PLUGIN_OK (resultado satisfactorio) si la contraseña es válida, o un código de error, tal como DB2SEC_PLUGIN_BADPWD, si la contraseña no es válida.

Esta API solo se invoca en el extremo cliente si el parámetro *authentication* tiene el valor CLIENT.

Sintaxis de la API de C:

```
SQL_API_RC SQL_API_FN db2secValidatePassword(
 const char *userid,
 db2int32  useridlen,
 const char *usernamespace,
 db2int32  usernamespacelen,
 db2int32  usernamespacetype,
 const char *password,
 db2int32  passwordlen,
 const char *newpassword,
 db2int32  newpasswordlen,
 const char *dbname,
 db2int32  dbnamelen,
  db2Uint32   connection_details,
  void **token,
  char **errormsg,
  db2int32  *errormsglen);
```

Entrada:

*const char *userid*
Es el ID de usuario cuya contraseña se debe verificar.

db2int32 useridlen
Es la longitud del ID de usuario.

*const char *password*
Es la contraseña que se debe verificar. DB2 cifra esta contraseña antes de pasarla al plug-in.

db2int32 passwordlen
Es la longitud de la contraseña proporcionada.

*const char *newpassword*
Es una contraseña nueva, si la contraseña se debe cambiar. Si no se solicita cambio de contraseña, el valor de este parámetro será NULL. Si el valor no es NULL, la función debe validar la contraseña antigua antes de cambiar a la nueva contraseña. El plug-in puede no satisfacer una petición de cambio de contraseña, pero si no satisface la petición, debe devolver inmediatamente DB2SEC_PLUGIN_CHANGEPASSWORD_NOTSUPPORTED sin validar la contraseña antigua.

db2int32 newpasswordlen
Es la longitud de la nueva contraseña.

*const char *dbname*
Es el nombre de la base de datos con la cual se establece conexión. La función puede hacer caso omiso de este campo, o puede devolver DB2SEC_PLUGIN_CONNECTIONREFUSED si desea limitar el acceso a determinadas bases de datos para usuarios que por lo demás tienen contraseñas validas.

db2int32 dbnamelen
Es la longitud del nombre de la base de datos.

db2int32 usernamespace
Es el espacio de nombres del cual se obtuvo el ID de usuario.

db2int32 usernamespacelen
Es la longitud del campo del espacio de nombres.

db2int32 usernamespacetype
Es el tipo del espacio de nombres. Los valores posibles son: DB2SEC_NAMESPACE_SAM_COMPATIBLE (corresponde a un estilo de nombre de usuario tal como torolab\miNombre") o DB2SEC_NAMESPACE_USER_PRINCIPAL (corresponde a un estilo de nombre de usuario tal como miNombre@torolab.ibm.com). Actualmente DB2 solo da soporte a DB2SEC_NAMESPACE_SAM_COMPATIBLE. Cuando no se conoce el ID de usuario, este campo se llena con DB2SEC_USER_NAMESPACE_UNDEFINED. Todas las sentencias DEFINE están situadas en `db2secPlugin.h`.

db2Uint32 connection_details
Es un campo de bits que actualmente consta de 2 campos:

- 1 bit indica si la conexión es local (utilizando IPC o conexión desde uno de los nodos contenidos en `db2nodes.cfg` del clúster EEE), o remota (a través de la red o bucle de retorno). Esto permite al plug-in decidir si los clientes situados en la misma máquina puede conectar con el servidor DB2 sin una contraseña. Actualmente, DB2 permite siempre las conexiones locales sin una contraseña para los clientes situados en la misma máquina (suponiendo que el cliente tenga privilegios de conexión).
- 1 bit indica si la fuente del ID de usuario es `db2secGetDefaultLoginContext` (fuente por omisión), o bien si el ID de

usuario se proporcionó explícitamente durante la conexión. Los valores de los bits están definidos en db2secPlugin.h:

```
#define DB2SEC_USERID_FROM_OS 0x00000001
```

DB2SEC_USERID_FROM_OS indica que el ID de usuario se obtiene del sistema operativo y no se proporciona explícitamente en la sentencia de conexión.

```
#define DB2SEC_CONNECTION_ISLOCAL 0x00000002
```

DB2SEC_CONNECTION_ISLOCAL denota una conexión local.

```
#define DB2SEC_VALIDATING_ON_SERVER_SIDE 0x0000004
```

DB2SEC_VALIDATING_ON_SERVER_SIDE indica si DB2 invoca el plug-in desde el extremo servidor para validar la contraseña.

El comportamiento por omisión de DB2 para una autenticación implícita es permitir la conexión sin realizar ninguna validación de contraseña. Sin embargo, el creador del plug-in puede opcionalmente inhabilitar la autenticación implícita mediante la devolución de un error DB2SEC_PLUGIN_BADPASSWORD.

*void **token*
Es un puntero que apunta a datos que se pueden pasar en llamadas subsiguientes de la API de plug-in (db2secGetAuthIDs, db2secGetGroupsForUser) durante esta conexión.

Salida:

*char **errormsg*
Es un puntero que indica la dirección de una cadena de caracteres ASCII asignada por el plug-in y que puede ser devuelta en este parámetro si la API no es efectiva.

*db2int32 *errormsglen*
Es puntero que apunta a un valor entero el cual indica la longitud de la cadena de mensaje de error contenida en *char **errormsg*.

Conceptos relacionados:

- "Plug-ins de seguridad" en la página 569
- "Las API de plug-in de seguridad" en la página 597

Información relacionada:

- "Las API del plug-in de autenticación por ID usuario/contraseña" en la página 608

db2secProcessServerPrincipalName - Procesar nombre de principal de servicio devuelto desde servidor

Esta función procesa el nombre de principal de servicio devuelto desde el servidor y pasa el nombre de principal al formato interno gss_name_t para utilizarlo con gss_init_sec_context(). Esta función también se invoca para procesar el nombre de principal de servicio que está catalogado en el directorio de base de datos en el caso de la autenticación Kerberos. Normalmente, esta conversión hace uso de la API gss_import_name(). Una vez establecido el contexto, se invoca gss_release_name() para liberar el objeto gss_name_t. Esta función devuelve

DB2SEC_PLUGIN_OK si gssName apunta a un nombre de GSS válido; devuelve un código de error DB2SEC_PLUGIN_BAD_PRINCIPAL_NAME si el nombre de principal no es válido.

Sintaxis de la API de C:

```
SQL_API_RC SQL_API_FN db2secProcessServerPrincipalName(
 const void *name,
 db2int32 nameLen,
 gss_name_t *gssName,
 char **errormsg,
 db2int32  *errormsglen);
```

Entrada:

*const void *name*
Es el nombre del principal de servicio, en formato GSS_C_NT_USER_NAME; por ejemplo, service/host@REALM.

db2int32 nameLen
Es la longitud del nombre del principal de servicio.

Salida:

*gss_name_t *gssName*
Es un puntero que apunta al nombre de principal de servicio, en el formato interno de GSS-API.

*char **errormsg*
Es un puntero que indica la dirección de una cadena de caracteres ASCII asignada por el plug-in y que puede ser devuelta en este parámetro si la API no es efectiva.

*db2int32 *errormsglen*
Es puntero que apunta a un valor entero el cual indica la longitud de la cadena de mensaje de error contenida en *char **errormsg*.

Conceptos relacionados:

- "Plug-ins de seguridad" en la página 569
- "Las API de plug-in de seguridad" en la página 597

Información relacionada:

- "Las API del plug-in de autenticación por ID usuario/contraseña" en la página 608

db2secFreeToken - Liberar memoria retenida por símbolo (token)

Esta función es invocada por DB2 cuando ya no necesita la memoria retenida por símbolo. El plug-in debe liberar la memoria.

Sintaxis de la API de C:

```
SQL_API_RC SQL_API_FN db2secFreeToken(
 void *token
 char **errormsg,
 db2int32  *errormsglen);
```

Entrada:

*void *token*
Es un puntero que apunta a la memoria que se debe liberar.

Salida:

*char **errormsg*
Es un puntero que indica la dirección de una cadena de caracteres ASCII asignada por el plug-in y que puede ser devuelta en este parámetro si la API no es efectiva.

*db2int32 *errormsglen*
Es puntero que apunta a un valor entero el cual indica la longitud de la cadena de mensaje de error contenida en *char **errormsg*.

Conceptos relacionados:

- "Plug-ins de seguridad" en la página 569
- "Las API de plug-in de seguridad" en la página 597

Información relacionada:

- "Las API del plug-in de autenticación por ID usuario/contraseña" en la página 608

db2secFreeInitInfo - Liberar recursos retenidos por db2secGenerateInitialCred()

Esta función libera los recursos asignados a `db2secGenerateInitialCred()`. Esto puede incluir, por ejemplo, descriptores de contexto de mecanismos subyacentes o una antememoria de credenciales de GSS-API.

Sintaxis de la API de C:

```
SQL_API_RC SQL_API_FN db2secFreeInitInfo(
 void *initinfo,
 char **errormsg,
 db2int32  *errormsglen);
```

Entrada:

*void *initinfo*
Es un puntero que apunta a datos que son opacos para DB2. El puntero se utiliza para mantener una lista de recursos que se asignan durante el proceso de generar el descriptor de credencial. Estos recursos se liberan invocando esta API.

Salida:

*char **errormsg*
Es un puntero que indica la dirección de una cadena de caracteres ASCII asignada por el plug-in y que puede ser devuelta en este parámetro si la API no es efectiva.

*db2int32 *errormsglen*
Es puntero que apunta a un valor entero el cual indica la longitud de la cadena de mensaje de error contenida en *char **errormsg*.

Conceptos relacionados:

- "Plug-ins de seguridad" en la página 569
- "Las API de plug-in de seguridad" en la página 597

Información relacionada:

- "Las API del plug-in de autenticación por ID usuario/contraseña" en la página 608

db2secServerAuthPluginInit - Inicializar el plug-in de autenticación del servidor

Esta es la función de inicialización correspondiente a la biblioteca que DB2 invoca inmediatamente después de cargar la biblioteca. El puntero `functions` se debe convertir a la estructura `server_auth_functions` apropiada para la versión de interfaz. En el caso de GSS-API, durante la inicialización, el plug-in proporciona el nombre de principal del servidor a la variable `serverPrincipalName` contenida en la estructura `gssapi_server_auth_functions` y proporciona el descriptor de credencial del servidor a la variable *serverCredHandle*. La función de limpieza `db2secServerAuthPluginTerm()` se encarga de liberar la memoria asignada para contener el nombre de principal y el descriptor de credencial.

Sintaxis de la API de C:

```
SQL_API_RC SQL_API_FN db2secServerAuthPluginInit(
 db2int32 version,
  void *server_fns,
  db2secGetConDetails *getConDetails_fn,
  db2secLogMessage *logMessage_fn,
  char **errormsg,
  db2int32  *errormsglen);
```

Entrada:

db2int32 version
: Es el número de versión más alto de la API al que DB2 da soporte actualmente.

*db2secGetConDetails *getConDetails_fn*
: Es un puntero que apunta a una función proporcionada por DB2. El plug-in puede invocar esta función en cualquiera de las demás API de autenticación para obtener detalles sobre la conexión de base de datos. Estos detalles incluyen información sobre el mecanismo de comunicación asociado a la conexión (tal como la dirección IP, en el caso de TCP/IP), que el creador del plug-in puede necesitar especificar al definir la autenticación. Por ejemplo, el plug-in podría rechazar una conexión para un determinado usuario, a menos que ese usuario se conecte desde una dirección IP determinada. El uso de esta llamada es opcional.

  Si la llamada se invoca en una situación en la que no interviene una conexión de base de datos, la función devuelve `DB2SEC_PLUGIN_NO_CON_DETAILS`; en otro caso, la función devuelve un 0 tras un resultado satisfactorio.

  El parámetro `getConDetails_fn` utiliza dos valores de entrada: un puntero que apunta a la estructura `db2sec_con_details`, y un número de versión que indica qué estructura `db2sec_con_details` se debe utilizar. El número de versión actual es 1. Después de la finalización satisfactoria de la función, la estructura `db2sec_con_details` contendrá los datos siguientes:

  - El protocolo utilizado para la conexión con el servidor. La lista de definiciones de protocolo se puede encontrar en el archivo `sqlenv.h` *lpar;SQL_PROTOCOL_*).

- La dirección TCP/IP de la conexión entrante con el servidor si el protocolo es TCP/IP.
- El nombre de la base de datos con la cual el cliente está intentando establecer conexión. Este nombre no se define para conexiones físicas de instancia.
- Un mapa de bits de información de conexión que contiene los mismos detalles que se describen en el parámetro `connection_details` de la API `db2secValidatePassword()`.

*db2secLogMessage *logMessage_fn*
Es un puntero que apunta a una función proporcionada por DB2. El plug-in puede invocar esta función para registrar en `db2diag.log` más información sobre errores para utilizarla con fines de depuración o información. El primer parámetro debe utilizar la sentencia DEFINE contenida en `db2secPlugin.h` y los dos últimos parámetros son el texto del mensaje y su longitud. Las sentencias DEFINE que se deben utilizar en el primer parámetro son:

```
#define DB2SEC_LOG_NONE       0   -  No hay registro de anotaciones
#define DB2SEC_LOG_CRITICAL   1   -  Encontrado error grave
#define DB2SEC_LOG_ERROR      2   -  Encontrado error
#define DB2SEC_LOG_WARNING    3   -  Aviso
#define DB2SEC_LOG_INFO       4   -  Informativo
```

Si utiliza la sentencia "define DB2SEC_LOG_INFO", el texto del mensaje solo aparecerá en el archivo `db2diag.log` si el parámetro de configuración *diaglevel* del gestor de bases de datos tiene el valor 4.

Salida:

*void *server_fns*
Es un puntero que apunta a memoria proporcionada por DB2 para una estructura `server_auth_functions`. En versiones futuras de DB2, pueden existir versiones diferentes de las API, por lo que esto se debe convertir a un puntero que apunte a la estructura `server_auth_functions` correspondiente a la versión de la API que el plug-in implementa.

Dentro de `server_auth_functions`, la variable `plugintype` debe tener uno de estos valores: DB2SEC_PLUGIN_TYPE_USERID_PASSWORD, DB2SEC_PLUGIN_TYPE_GSSAPI o DB2SEC_PLUGIN_TYPE_KERBEROS. Podrán definirse otros valores en versiones futuras de la API.

*char **errormsg*
Es un puntero que indica la dirección de una cadena de caracteres ASCII asignada por el plug-in y que puede ser devuelta en este parámetro si la API no es efectiva.

*db2int32 *errormsglen*
Es puntero que apunta a un valor entero el cual indica la longitud de la cadena de mensaje de error contenida en *char **errormsg*.

Conceptos relacionados:

- "Plug-ins de seguridad" en la página 569
- "Las API de plug-in de seguridad" en la página 597

Información relacionada:

- "diaglevel - Diagnostic error capture level configuration parameter" en la publicación *Administration Guide: Performance*

- "Las API del plug-in de autenticación por ID usuario/contraseña" en la página 608

db2secServerAuthPluginTerm - Depurar recursos de plug-in de autenticación de servidor

DB2 invoca esta función justo antes de descargar el plug-in. La función debe realizar una depuración apropiada de los recursos que estén retenidos por la biblioteca del plug-in, tal como liberar la memoria asignada por el plug-in, cerrar los archivos que estén todavía abiertos y cerrar las conexiones de red. El plug-in realiza un seguimiento de estos recursos a fin de liberarlos.

Sintaxis de la API de C:

```
SQL_API_RC SQL_API_FN db2secServerAuthPluginTerm(char **errormsg,
 db2int32  *errormsglen);
```

Salida:

*char **errormsg*
Es un puntero que indica la dirección de una cadena de caracteres ASCII asignada por el plug-in y que puede ser devuelta en este parámetro si la API no es efectiva.

*db2int32 *errormsglen*
Es puntero que apunta a un valor entero el cual indica la longitud de la cadena de mensaje de error contenida en *char **errormsg*.

Conceptos relacionados:

- "Plug-ins de seguridad" en la página 569
- "Las API de plug-in de seguridad" en la página 597

Información relacionada:

- "Las API del plug-in de autenticación por ID usuario/contraseña" en la página 608

db2secGetAuthIDs - Obtener los ID de autenticación

Esta función devuelve un ID de autorización de SQL para un usuario autenticado. La función se invoca durante las conexiones a una base de datos para los métodos de autenticación basados en un ID de usuario/contraseña y para los basados en GSS-API.

Sintaxis de la API de C:

```
SQL_API_RC SQL_API_FN db2secGetAuthIDs(
 const char *userid,
 db2int32  useridlen,
 const char *usernamespace,
 db2int32  usernamespacelen,
 db2int32  usernamespacetype,
 const char *dbname,
 db2int32  dbnamelen,
 void **token,
 char SystemAuthID[DB2SEC_MAX_AUTHID_LENGTH],
 db2int32  *SystemAuthIDlen,
 char InitialSessionAuthID[DB2SEC_MAX_AUTHID_LENGTH],
 db2int32  *InitialSessionAuthIDlen,
 char username[DB2SEC_MAX_USERID_LENGTH],
```

```
db2int32  *usernamelen,
db2int32 *initsessionidtype,
char **errormsg,
db2int32  *errormsglen);
```

Entrada:

*const char * userid*
: Es el usuario autenticado. Este campo estará en blanco para GSS-API.

db2int32 useridlen
: Es la longitud del ID de usuario.

*void **token*
: Son datos que el plug-in puede pasar a la llamada `db2secGetGroupsForUser`. Para GSS-API, esto es un descriptor de contexto (`gss_ctx_id_t`). Normalmente, esto es un parámetro solo de entrada y su valor se obtiene de `db2secValidatePassword`. También puede ser un parámetro de salida cuando la autenticación se realiza en el cliente y por tanto no se invoca `db2secValidatePassword`.

*const char *dbname*
: Es el nombre de la base de datos con la cual se establece conexión. El plug-in puede no tener en cuenta este campo o devolver unos ID de autorización diferentes cuando un mismo usuario se conecta a bases de datos diferentes.

db2int32 dbnamelen
: Es la longitud del nombre de la base de datos.

*const char *usernamespace*
: Es el espacio de nombres del cual se obtuvo el ID de usuario.

db2int32 usernamespacelen
: Es la longitud del campo del espacio de nombres.

db2int32 usernamespacetype
: Es aplicable lo descrito más arriba. En la versión actual de DB2, el único tipo soportado de espacio de nombres es DB2SEC_NAMESPACE_SAM_COMPATIBLE.

Salida:

char SystemAuthID[DB2SEC_MAX_AUTHID_LENGTH]
: El ID de autorización del sistema corresponde al ID del usuario autenticado. El tamaño es 255, pero DB2 solo puede actualmente utilizar hasta 30.

*db2int32 *SystemAuthiIDlen*
: Es la longitud del ID de autorización del sistema (`SystemAuthId`) devuelto.

char InitialSessionAuthid[DB2SEC_MAX_AUTHID_LENGTH]
: Es el ID de autorización utilizado para la sesión de conexión. Normalmente esto es igual a `SystemAuthID`, pero puede ser diferente en determinados casos, por ejemplo, cuando se emite una sentencia para definir la autorización de la sesión. El tamaño es 255, pero DB2 solo puede actualmente utilizar hasta 30.

*db2int32 *InitialSessionAuthidlen*
: Es la longitud del ID de autorización de sesión inicial (`InitialSessionAuthID`) devuelto.

char username[DB2SEC_MAX_USERID_LENGTH]
Es un nombre de usuario que corresponde al usuario autenticado e ID de autorización. Este nombre solo se utiliza con fines de auditoría y se registra en el campo "ID de usuario". Si el plug-in no rellena este campo, DB2 lo copiará a partir del ID de usuario.

*db2int32 *usernamelen*
Es la longitud del ID de usuario devuelto.

*db2int32 *initsessionidtype*
Es un tipo de ID de autorización de sesión, que indica si `InitialSessionAuthid` es un rol o un ID de autorización. El plug-in devuelve uno de los valores siguientes (definidos en `db2secPlugin.h`): `DB2SEC_ID_TYPE_AUTHID` (0) o `DB2SEC_ID_TYPE_ROLE` (1). Actualmente, DB2 solo da soporte al ID de autorización (DB2SEC_ID_TYPE_AUTHID).

*char **errormsg*
Es un puntero que indica la dirección de una cadena de caracteres ASCII asignada por el plug-in y que puede ser devuelta en este parámetro si la API no es efectiva.

*db2int32 *errormsglen*
Es puntero que apunta a un valor entero el cual indica la longitud de la cadena de mensaje de error contenida en *char **errormsg*.

Conceptos relacionados:

- "Plug-ins de seguridad" en la página 569
- "Las API de plug-in de seguridad" en la página 597

Información relacionada:

- "Las API del plug-in de autenticación por ID usuario/contraseña" en la página 608

db2secDoesAuthIDExist - Comprobar si existe el ID de autenticación

Esta función determina si el ID de autorización representa un usuario individual (por ejemplo, si la función puede o no correlacionar el ID de autorización con un ID de usuario externo). Esta función devuelve DB2SEC_PLUGIN_OK si el resultado es satisfactorio, es decir, si el ID de autorización es válido; devuelve DB2SEC_PLUGIN_INVALID_USERORGROUP si no es válido o DB2SEC_PLUGIN_USERSTATUSNOTKNOWN si no se puede determinar su existencia.

Sintaxis de la API de C:

```
SQL_API_RC SQL_API_FN db2secDoesAuthIDExist(
 const char *authid,
 db2int32  authidlen,
 const char *errormsg,
 db2int32  *errormsglen);
```

Entrada:

*const char *authid*
Es el ID de autorización que se debe validar. Está escrito en mayúsculas y sin blancos de cola.

db2int32 authidlen
Es la longitud del ID de autorización.

Salida:

*char **errormsg*
Es un puntero que indica la dirección de una cadena de caracteres ASCII asignada por el plug-in y que puede ser devuelta en este parámetro si la API no es efectiva.

*db2int32 *errormsglen*
Es puntero que apunta a un valor entero el cual indica la longitud de la cadena de mensaje de error contenida en *char **errormsg*.

Conceptos relacionados:

- "Plug-ins de seguridad" en la página 569
- "Las API de plug-in de seguridad" en la página 597

Información relacionada:

- "Las API del plug-in de autenticación por ID usuario/contraseña" en la página 608

Las API de plug-in de GSS-API

Las API y definiciones necesarias para los plug-ins de autenticación de GSS-API

Este tema muestra una lista completa de las API de GSS necesarias para la interfaz de plug-ins de seguridad de DB2. Las API soportadas siguen estas especificaciones: *Generic Security Service Application Program Interface, Versión 2* (IETF RFC2743) y *Generic Security Service API Versión 2: C-Bindings* (IETF RFC2744). Antes de implementar un plug-in basado en GSS-API, debe tener un conocimiento completo de esas especificaciones.

Tabla 84. Las API y definiciones necesarias para los plug-ins de autenticación de GSS-API

Nombre		Descripción
Las API del extremo cliente		
	`gss_init_sec_context`	Iniciar un contexto de seguridad con una aplicación homóloga
Las API del extremo servidor		
	`gss_accept_sec_context`	Aceptar un contexto de seguridad iniciado por una aplicación homóloga.
	`gss_display_name`	Convertir a texto un nombre con formato interno.
Las API comunes		
	`gss_delete_sec_context`	Suprimir un contexto de seguridad establecido.
	`gss_display_status`	Obtener el mensaje de error de texto asociado a un código de estado de GSS-API.
	`gss_release_buffer`	Suprimir un almacenamiento intermedio.
	`gss_release_cred`	Liberar las estructuras de datos locales asociadas a una credencial de GSS-API.
	`gss_release_name`	Suprimir un nombre con formato interno.

Tabla 84. Las API y definiciones necesarias para los plug-ins de autenticación de GSS-API (continuación)

Nombre		Descripción
Definiciones necesarias		
	GSS_C_DELEG_FLAG	Se ha solicitado delegación.
	GSS_C_EMPTY_BUFFER	Significa que gss_buffer_desc no contiene ningún dato.
	GSS_C_GSS_CODE	Denota un código de estado principal de GSS.
	GSS_C_INDEFINITE	Indica que el mecanismo no da soporte a la caducidad de contexto.
	GSS_C_MECH_CODE	Denota un código de estado secundario de GSS.
	GSS_C_MUTUAL_FLAG	Se ha solicitado autenticación mutua.
	GSS_C_NO_BUFFER	Denota que la variable gss_buffer_t no apunta a una estructura gss_buffer_desc válida.
	GSS_C_NO_CHANNEL_BINDINGS	No existen vinculaciones de canales de comunicación.
	GSS_C_NO_CONTEXT	Denota que la variable gss_ctx_id_t no apunta a un contexto válido.
	GSS_C_NO_CREDENTIAL	Denota que la variable gss_cred_id_t no apunta a un descriptor de credencial válido.
	GSS_C_NO_NAME	Denota que la variable gss_name_t no apunta a un nombre interno válido.
	GSS_C_NO_OID	Utilizar el mecanismo de autenticación por omisión.
	GSS_C_NULL_OID_SET	Utilizar el mecanismo por omisión.
	GSS_S_COMPLETE	La API se ejecutó satisfactoriamente.
	GSS_S_CONTINUE_NEEDED	El proceso no ha finalizado y se debe invocar de nuevo la API utilizando el símbolo de respuesta recibido de la entidad homóloga.

Conceptos relacionados:

- "Plug-ins de seguridad" en la página 569
- "Las API de plug-in de seguridad" en la página 597

Información relacionada:

- "Restricciones para los plug-in de autenticación de GSS-API" en la página 633

Restricciones para los plug-in de autenticación de GSS-API

A continuación sigue una lista de restricciones para los plug-in de autenticación de GSS-API.

- Se considera siempre que se utiliza el mecanismo de seguridad por omisión, por lo que no se hace ninguna consideración respecto a OID.
- Los únicos servicios de GSS solicitados en gss_init_sec_context() son la autenticación mutua y la delegación. DB2 solicita siempre un certificado para la delegación, pero no utilizará actualmente ese certificado para generar un nuevo certificado.
- Solo se solicitará el tiempo de contexto por omisión.
- No se envían señales de contexto de gss_delete_sec_context() desde el cliente al servidor y viceversa.
- El anonimato no está soportado.

- La vinculación de canal no está soportada.
- Si las credenciales iniciales caducan, DB2 no las renueva automáticamente.
- La especificación GSS-API establece que aunque fallen las funciones `gss_init_sec_context()` o `gss_accept_sec_context()`, ambas funciones deben devolver una señal para enviar al nodo interlocutor. Sin embargo, debido a limitaciones de DRDA, DB2 solo puede enviar una señal si `gss_init_sec_context()` falla y genera una señal en la primera llamada.

Conceptos relacionados:

- "Plug-ins de seguridad" en la página 569
- "Las API de plug-in de seguridad" en la página 597

Información relacionada:

- "Las API y definiciones necesarias para los plug-ins de autenticación de GSS-API" en la página 632

Creación de versiones de la API de plug-in de seguridad

Debido a que es posible que releases futuros de DB2® necesiten versiones diferentes de las API del plug-in de seguridad, DB2 da soporte a la numeración de versiones de las API. Estos números de versión son valores enteros que comienzan con el 1 para DB2 UDB Versión 8.2. El número de versión que DB2 pasa a las API del plug-in de seguridad es el número de versión más alto de la API que DB2 puede soportar, y corresponde a un número de versión de la estructura. Si el plug-in puede soportar un versión de API más alta, debe devolver punteros de función para la versión que DB2 ha solicitado. Si el plug-in solo puede dar soporte a una versión inferior de la API, debe proporcionar punteros de función para esa versión inferior. En ambos casos, las API de plug-in de seguridad deben devolver el número de versión de la API soportada en el campo de versión de la estructura de función.

Para DB2, los números de versión de los plug-ins de seguridad solo cambiarán cuando sea necesario. Por ejemplo, cuando se produzcan cambios en los parámetros de las API. Los números de versión no cambiarán automáticamente con los números de release de DB2.

Conceptos relacionados:

- "Plug-ins de seguridad" en la página 569
- "Las API de plug-in de seguridad" en la página 597

Parte 7. Conceptos generales sobre aplicaciones DB2

Capítulo 29. Soporte de idiomas nacionales

Visión general del orden de clasificación 637
Ordenes de clasificación 637
Comparaciones de caracteres basadas en ordenes de clasificación 639
Comparaciones que no dependen de mayúsculas y minúsculas mediante la función TRANSLATE 640
Diferencias entre los órdenes de clasificación de EBCDIC y ASCII 641
Orden de clasificación especificado al crear una base de datos 642
Órdenes de clasificación de ejemplo 644
Páginas de códigos y entornos locales 645
Derivación de valores de página de códigos . . 645
Derivación de entornos locales en programas de aplicación 645
Cómo DB2 deriva entornos locales 646
Consideraciones sobre aplicaciones 646
Consideraciones sobre el soporte de idiomas nacionales y sobre el desarrollo de aplicaciones . 646
Soporte de idiomas nacionales y sentencias de SQL 648
Rutinas remotas 649
Consideraciones sobre nombres de paquetes en entornos de páginas de códigos combinadas . . 649
Página de códigos activa para precompilación y vinculación 650
Página de códigos activa para la ejecución de aplicaciones 650
Conversión de caracteres entre páginas de códigos diferentes 650
Cuándo se produce conversión de la página de códigos 650
Sustitución de caracteres durante conversiones de páginas de códigos 651
Conversiones de páginas de códigos soportadas 652
Factor de expansión de conversión de página de códigos 653
Juegos de caracteres DBCS 654
Juegos de caracteres de Extended UNIX Code (EUC) 654
Programas CLI, ODBC, JDBC y SQLJ en un entorno DBCS 655
Consideraciones sobre los juegos de códigos EUC y UCS-2 de japonés y chino tradicional 656
Consideraciones sobre los juegos de códigos EUC y UCS-2 de japonés y chino tradicional . . 656
Consideraciones sobre las bases de datos y los clientes de doble byte con EUC mixtos 658
Consideraciones sobre la conversión de caracteres para usuarios de chino tradicional . . 658
Datos gráficos en aplicaciones EUC en japonés o chino tradicional 659
Desarrollo de aplicaciones en situaciones de páginas de códigos distintas 660
Validación de parámetros basada en cliente en un entorno de juegos de códigos mixtos . . . 663
Sentencia DESCRIBE en entornos de juegos de códigos mixtos 664
Datos de longitud fija y de longitud variable en entornos de juegos de códigos mixtos 666
Desbordamiento de longitud de serie de conversión de página de códigos en entornos de juegos de códigos mixtos 666
Aplicaciones conectadas a bases de datos Unicode 668

Visión general del orden de clasificación

Las secciones siguientes describen órdenes de clasificación y cómo se realizan las comparaciones de caracteres.

Ordenes de clasificación

El gestor de bases de datos compara datos de tipo carácter utilizando un *orden de clasificación*. Es una clasificación para conjuntos de caracteres que determina si un determinado carácter tiene una posición de clasificación superior, inferior o igual que otro.

Nota: Los datos de tipo carácter definidos con el atributo FOR BIT DATA y los datos BLOB se clasifican utilizando el orden de clasificación binaria.

Por ejemplo, se puede utilizar un orden de clasificación para indicar que las versiones en minúsculas y en mayúsculas de un determinado carácter se tienen que clasificar de igual forma.

El gestor de bases de datos permite que se creen bases de datos con órdenes de clasificación personalizados. Las secciones siguientes le ayudan a determinar y a implantar un determinado orden de clasificación para una base de datos.

Cada carácter de un solo byte de una base de datos se representa de forma interna como un número exclusivo comprendido entre 0 y 255 (en notación hexadecimal, entre X'00' y X'FF'). Se hace referencia a este número como *punto de código* del carácter; la asignación de números a caracteres en un conjunto se denomina de forma colectiva *página de códigos*. Un orden de clasificación es una correlación entre el punto de código y la posición deseada de cada carácter en un orden clasificado. El valor numérico de la posición se denomina *peso* del carácter en el orden de clasificación. En el orden de clasificación más sencillo, los pesos son idénticos a los puntos de código. Esto se denomina el *orden de identidad*.

Por ejemplo, supongamos que los caracteres B y b tienen los puntos de código X'42' y X'62' respectivamente. Si (según la tabla de orden de clasificación), ambos tienen un peso de clasificación de X'42' (B), se clasifican igual. Si el peso de clasificación correspondiente a B es X'9E' y el correspondiente a b es X'9D', b se clasificará antes que B. Ta tabla de orden de clasificación especifica el peso de cada carácter. La tabla es distinta de una página de códigos, la cual especifica el elemento de código de cada carácter.

Supongamos que tenemos el siguiente ejemplo. Los caracteres ASCII comprendidos entre A y Z se representan mediante los valores comprendidos entre X'41' y X'5A'. Para describir un orden de clasificación en el que estos caracteres se clasifican de forma consecutiva (sin caracteres intermedios), puede escribir X'41', X'42', ... X'59', X'5A'.

El valor hexadecimal de un carácter de varios bytes también se utiliza como el peso. Por ejemplo, supongamos que los puntos de código correspondientes a los caracteres de doble byte A y B son X'8260' y X'8261' respectivamente; se utilizan los pesos de clasificación correspondientes a X'82', X'60' y X'61' para clasificar estos dos caracteres se acuerdo con sus puntos de código.

No hace falta que los pesos de un orden de clasificación sean exclusivos. Por ejemplo, podría asignar a letras mayúsculas y a sus equivalentes minúsculas el mismo peso.

La especificación de un orden de clasificación se puede simplificar si el orden de clasificación proporciona pesos para los 256 puntos de código. El peso de cada carácter se puede determinar mediante el punto de código del carácter.

En todos los casos, DB2 Universal Database™ (DB2 UDB) utiliza la tabla de clasificación que se especificó al crear la base de datos. Si desea que los caracteres de varios bytes se clasifiquen del modo en que aparecen en su tabla de elementos de código, debe especificar IDENTITY como el orden de clasificación al crear la base de datos.

Una vez definido un orden de clasificación, todas las futuras comparaciones de caracteres correspondientes a dicha base de datos se realizará con este orden de clasificación. Excepto para los datos definidos como FOR BIT DATA o datos BLOB, el orden de clasificación se utilizará para todas las comparaciones de SQL y cláusulas ORDER BY y también al configurar índices y estadísticas.

Se pueden producir problemas en los siguientes casos:

- Una aplicación fusiona datos clasificados de una base de datos con datos de aplicación que se clasificaron utilizando otro orden de clasificación.
- Una aplicación fusiona datos clasificados de una base de datos con datos clasificados de otra, pero las bases de datos tienen distintos órdenes de clasificación.
- Una aplicación realiza supuestos sobre datos clasificados que no resultan ciertos para el orden de clasificación relevante. Por ejemplo, que los números se clasifican por debajo que los caracteres alfabéticos puede resultar o no cierto para un determinado orden de clasificación.

Un punto final a tener en cuenta es que los resultados de cualquier clasificación basada en una comparación directa de puntos de código de caracteres sólo coincidirá con los resultados de consultas que estén clasificados utilizando un orden de clasificación de identidad.

Conceptos relacionados:

- "Conversión de caracteres" en la publicación *Consulta de SQL, Volumen 1*
- "Unicode implementation in DB2 Universal Database" en la publicación *Administration Guide: Planning*
- "Comparaciones de caracteres basadas en ordenes de clasificación" en la página 639

Comparaciones de caracteres basadas en ordenes de clasificación

Una vez establecido un orden de clasificación para una base de datos mediante SYSTEM, NLSCHAR, COMPATIBILITY, o mediante una definición del usuario, la comparación de caracteres se realiza comparando los pesos de dos caracteres, en lugar de comparar directamente los valores de sus elementos de código.

Si se utilizan pesos que no son exclusivos, es posible que caracteres que no son idénticos se comparen y queden iguales. Por este motivo, la comparación de series se puede convertir en un proceso de dos fases:

1. Comparar los caracteres de cada serie según sus pesos.
2. Si el paso 2 da como resultado igualdad, comparar los caracteres de cada serie según sus valores de punto de código.

Si el orden de clasificación contiene 256 pesos exclusivos, sólo hay que llevar a cabo el primer paso. Si el orden de clasificación es el orden de identidad, sólo hay que llevar a cabo el segundo paso. En cualquier caso, hay una mejora del rendimiento. Para las bases de datos Unicode, si la opción de clasificación es SYSTEM o IDENTITY, el orden de clasificación será IDENTITY y solo se ejecuta el segundo paso.

Para una base de datos Unicode con la opción de clasificación SQL_CS_IDENTITY_16BIT, los datos CHAR o VARCHAR de la base de datos se clasificarán de acuerdo con su orden binario CESU-8 en lugar de seguir el orden binario UTF-8. CESU-8 significa *Compatibility Encoding Scheme for UTF-16: 8-Bit*, y está definido en la publicación *Unicode Technical Report #26*, que se puede consultar en el sitio Web de Unicode Consortium (www.unicode.org). CESU-8 es idéntico en valor binario a UTF-8, excepto para los caracteres complementarios de Unicode, es decir, los caracteres que están definidos fuera de Basic Multilingual Plane de 16 bits (BMP o Plane 0). En la codificación UTF-8, un carácter complementario se representa mediante una secuencia de 4 bytes, pero el mismo carácter en CESU-8

necesita dos secuencias de 3 bytes. En una base de datos Unicode, los datos de tipo carácter se guardan según la codificación UTF-8 y los datos gráficos según la codificación UCS-2. Para la clasificación SQL_CS_NONE, los caracteres no complementarios en UTF-8 y UCS-2 tienen la misma clasificación binaria, pero los caracteres complementarios en UTF-8 se clasifican de forma distinta respecto de los mismos caracteres en UCS-2.

Para las bases de datos Unicode cuya opción de clasificación es de tipo UCA (Unicode Collation Algorithm), los caracteres que no tienen el mismo valor binario pero cuyo significado es el mismo, serán considerados como iguales en la comparación. Por este motivo, la comparación de series de caracteres se puede convertir en un proceso de dos fases:

1. Comparar los caracteres de cada cadena de acuerdo con el algoritmo especificado en la norma *Unicode Technical Standard #10*, disponible en el sitio Web de Unicode Technical Consortium (www.unicode.org).
2. Si el paso 1 produce un resultado de igualdad, comparar los caracteres de cada cadena según los valores del elemento de código respectivo.

Conceptos relacionados:

- "Conversión de caracteres" en la publicación *Consulta de SQL, Volumen 1*

Comparaciones que no dependen de mayúsculas y minúsculas mediante la función TRANSLATE

Para realizar comparaciones de caracteres que no dependen de mayúsculas y minúsculas, puede utilizar la función TRANSLATE para seleccionar y comparar datos de columnas de mayúsculas y minúsculas combinadas, convirtiéndolos todos a mayúsculas (sólo para la comparación). Supongamos que tenemos los datos siguientes:

```
Abel
abels
ABEL
abel
ab
Ab
```

La siguiente sentencia SELECT:

```
SELECT c1 FROM T1 WHERE TRANSLATE(c1) LIKE 'AB%'
```

devuelve

```
ab
Ab
abel
Abel
ABEL
abels
```

También podría especificar la siguiente sentencia SELECT al crear la vista ″v1″, realizar todas las comparaciones contra la vista en mayúsculas y solicitar operaciones INSERT de la tabla en mayúsculas y minúsculas combinadas:

```
CREATE VIEW v1 AS SELECT TRANSLATE(c1) FROM T1
```

A nivel de base de datos, puede definir el orden de clasificación como parte de la API **sqlecrea**. - Crear base de datos. Esto le permite decidir si ″a″ se tiene que procesar antes que ″A″, si ″A″ se tiene que procesar después que ″a″ o si se tienen que procesar con el mismo peso. Esto las convertiría en iguales si se realiza la

clasificación mediante la cláusula ORDER BY. ″A″ siempre irá antes que ″a″, porque son equivalentes en cualquier sentido. La única base por la que clasificar es el valor hexadecimal.

Por lo tanto

```
SELECT c1 FROM T1 WHERE c1 LIKE 'ab%'
```

devuelve

```
ab
abel
abels
```

y

```
SELECT c1 FROM T1 WHERE c1 LIKE 'A%'
```

devuelve

```
Abel
Ab
ABEL
```

La sentencia siguiente

```
SELECT c1 FROM T1 ORDER BY c1
```

devuelve

```
ab
Ab
abel
Abel
ABEL
abels
```

Por lo tanto, quizás desee tener en cuenta la posibilidad de utilizar la función escalar TRANSLATE(), además de **sqlecrea**. Tenga en cuenta que sólo puede especificar un orden de clasificación mediante **sqlecrea**. No puede especificar un orden de clasificación desde el procesador de línea de mandatos (CLP).

También puede utilizar la función UCASE del siguiente modo, pero tenga en cuenta que DB2® realiza una exploración de tabla en lugar de utilizar un índice para esta operación select:

```
SELECT * FROM EMP WHERE UCASE(JOB) = 'NURSE'
```

Información relacionada:

- "Función escalar TRANSLATE" en la publicación *Consulta de SQL, Volumen 1*
- "Función escalar UCASE o UPPER" en la publicación *Consulta de SQL, Volumen 1*
- "sqlecrea - Create Database" en la publicación *Administrative API Reference*

Diferencias entre los órdenes de clasificación de EBCDIC y ASCII

El orden en que se clasifican los datos de una base de datos depende del orden de clasificación definido para la base de datos. Por ejemplo, supongamos que la base de datos A utiliza el orden de clasificación por omisión de la página de códigos EBCDIC y que la base de datos B utiliza el orden de clasificación por omisión de la

página de códigos ASCII. Los órdenes de clasificación de estas dos bases de datos diferirían, tal como se muestra en el siguiente ejemplo:

```
SELECT.....
  ORDER BY COL2

EBCDIC-Based Sort        ASCII-Based Sort

COL2                     COL2
----                     ----
V1G                      7AB
Y2W                      V1G
7AB                      Y2W
```

Figura 64. Ejemplo de cómo difiere un orden de clasificación basado en EBCDIC de un orden de clasificación basado en ASCII

De forma similar, las comparaciones de caracteres de una base de datos dependen del orden de clasificación definido para dicha base de datos. De este modo, si la base de datos A utiliza el orden de clasificación por omisión de la página de códigos EBCDIC y la base de datos B utiliza el orden de clasificación por omisión de la página de códigos ASCII, los resultados de comparaciones de caracteres en las dos bases de datos diferirían. La diferencia es la siguiente:

```
SELECT.....
  WHERE COL2 > 'TT3'

EBCDIC-Based Results     ASCII-Based Results

COL2                     COL2
----                     ----
TW4                      TW4
X72                      X72
39G
```

Figura 65. Ejemplo de cómo una comparación de caracteres en un orden de clasificación basado en EBCDIC difiere de una comparación de caracteres en un orden de clasificación basado en ASCII

Si va a crear una base de datos federada, considere la posibilidad de especificar que el orden de clasificación coincida con el orden de clasificación de la fuente de datos. Este enfoque maximizará las oportunidades de "ordenación en pila" y posiblemente aumentará el rendimiento de las consultas.

Conceptos relacionados:

- "Guidelines for analyzing where a federated query is evaluated" en la publicación *Administration Guide: Performance*

Orden de clasificación especificado al crear una base de datos

El orden de clasificación correspondiente a una base de datos se especifica en el momento de creación de la base de datos. Una vez creada la base de datos, el orden de clasificación no se puede cambiar.

La API CREATE DATABASE acepta una estructura de datos denominada Bloque descriptor de base de datos (SQLEDBDESC). Puede definir su propio orden de clasificación dentro de esta estructura.

Nota: Sólo puede definir su propio orden de clasificación para una base de datos de un solo byte.

Para especificar un orden de clasificación para una base de datos:
- Pase la estructura SQLEDBDESC que desee, o
- Pase un puntero NULL. Se utiliza el orden de clasificación del sistema operativo (basado en la página de códigos y la página de país/región actuales). Esto equivale a especificar para SQLDBCSS el valor SQL_CS_SYSTEM (0).

La estructura SQLEDBDESC contiene:

SQLDBCSS Un entero de 4 bytes que indica la fuente del orden de clasificación de la base de datos. Los valores válidos son:

SQL_CS_SYSTEM
Se utiliza el orden de clasificación del sistema operativo (basado en la página de códigos y la página de país/región actuales).

SQL_CS_SYSTEM_NLSCHAR
Orden de clasificación del usuario que utiliza la versión NLS de rutinas de comparación para tipos de caracteres

SQL_CS_IDENTITY_16BIT
Una base de datos Unicode se puede crear con la opción de clasificación SQL_CS_IDENTITY_16BIT. SQL_CS_IDENTITY_16BIT difiere de la opción de clasificación por omisión SQL_CS_NONE en que los datos CHAR o VARCHAR de la base de datos Unicode se clasificarán utilizando el orden binario CESU-8 en lugar del orden binario UTF-8. CESU-8 significa *Compatibility Encoding Scheme for UTF-16: 8-Bit*, y está definido en la publicación *Unicode Technical Report #26*, que se puede consultar en el sitio Web de Unicode Consortium (www.unicode.org). CESU-8 es idéntico en binario a UTF-8, excepto los caracteres complementarios de Unicode, es decir, los caracteres que están definidos fuera de Basic Multilingual Plane de 16 bits (BMP o Plane 0). En codificación UTF-8, un carácter complementario se representa mediante una secuencia de 4 bytes, pero el mismo carácter en CESU-8 necesita dos secuencias de 3 bytes. En una base de datos Unicode, los datos de tipo carácter se guardan según la codificación UTF-8 y los datos gráficos según la codificación UCS-2. Para la clasificación SQL_CS_NONE, los caracteres no complementarios en UTF-8 y UCS-2 tienen la misma clasificación binaria, pero los caracteres complementarios en UTF-8 se clasifican de forma distinta a los mismos caracteres en UCS-2. SQL_CS_IDENTITY_16BIT asegura que todos los caracteres, complementarios y no complementarios, de una base de datos Unicode DB2® tienen la misma clasificación binaria.

SQL_CS_UCA_NO
Una base de datos Unicode se puede crear con la opción de clasificación SQL_CS_UCA400_NO. SQL_CS_UCA400_NO especifica el orden de clasificación de UCA (Unicode Collation Algorithm) basado en el Estándar Unicode versión 4.00, con la normalización habilitada implícitamente. Puede encontrar detalles sobre UCA en la publicación *Unicode Technical Standard #10*, que se encuentra en el sitio Web de Unicode Consortium (www.unicode.org).

SQL_CS_UCA_LTH
Una base de datos Unicode se puede crear con la opción de clasificación SQL_CS_UCA400_LTH. SQL_CS_UCA400_LTH especifica el orden de clasificación UCA (Unicode Collation Algorithm) basado en el Estándar Unicode versión 4.00, pero clasifica los caracteres Thai de acuerdo con el orden definido por el diccionario Royal Thai Dictionary. Puede encontrar detalles sobre UCA en la publicación *Unicode Technical Standard #10*, que se encuentra en el sitio Web de Unicode Consortium (www.unicode.org).

SQL_CS_USER
El orden de clasificación se especifica mediante el valor del campo SQLDBUDC.

SQL_CS_NONE
El orden de clasificación es la clasificación de identidad. Las series se comparan byte a byte, empezando por el primer byte, utilizando una sencilla comparación de puntos de código.

Nota: Estas constantes se definen en el archivo include SQLENV.

SQLDBUDC Un campo de 256 bytes. El byte número *n* contiene el peso de clasificación del carácter número *n* de la página de códigos de la base de datos. Si SQLDBCSS no es igual a SQL_CS_USER, este campo se pasa por alto.

Información relacionada:

- "sqlecrea - Create Database" en la publicación *Administrative API Reference*

Órdenes de clasificación de ejemplo

Se proporcionan varios órdenes de clasificación de ejemplo (como archivos include) para facilitar la creación de bases de datos utilizando los órdenes de clasificación EBCDIC en lugar del orden de clasificación por omisión de la estación de trabajo.

Los órdenes de clasificación de estos archivos include se pueden especificar en el campo SQLDBUDC de la estructura SQLEDBDESC. También se pueden utilizar como modelos para la construcción de otros órdenes de clasificación.

Hay archivos include que contienen órdenes de clasificación disponibles para los siguientes lenguajes principales:

- C/C++
- COBOL
- FORTRAN

Información relacionada:

- "Archivos include para C y C++" en la página 147
- "Archivos include para COBOL" en la página 194
- "Archivos include para FORTRAN" en la página 216

Páginas de códigos y entornos locales

Las secciones siguientes describen páginas de códigos y cómo se obtienen páginas de códigos y entornos locales.

Derivación de valores de página de códigos

La *página de códigos de la aplicación* se deriva del entorno activo cuando se realiza la conexión con la base de datos.Si la variable de registro DB2CODEPAGE está definida, se toma este valor como página de códigos de la aplicación. Sin embargo, no es necesario definir la variable de registro DB2CODEPAGE porque DB2® determinará el valor adecuado de página de códigos a partir del sistema operativo. Si se define para la variable de registro DB2CODEPAGE un valor incorrecto se pueden producir resultados inesperados.

La *página de códigos de la base de datos* se deriva del valor especificado (de forma explícita o por omisión) en el momento en que se crea la base de datos.Por ejemplo, lo siguiente define cómo se determina el *entorno activo* en distintos sistemas operativos:

UNIX® En sistemas operativos basados en UNIX, el entorno activo se determina a partir del valor de entorno local (locale) que incluye información sobre idioma, territorio y juego de códigos.

Sistemas operativos Windows®
Para todos los sistemas operativos Windows, si la variable de entorno DB2CODEPAGE no está definida, la página de códigos se deriva del valor de página de códigos ANSI del Registro.

La *página de códigos de sección* se obtiene de las tablas utilizadas en una sentencia de SQL. Si las tablas están definidas implícita o explícitamente con CCSID ASCII, la página de códigos de sección es la misma que la página de códigos de la base de datos. Si las tablas están definidas con CCSID UNICODE, la página de códigos de sección es la página de códigos Unicode.

Información relacionada:

- "Supported territory codes and code pages" en la publicación *Administration Guide: Planning*

Derivación de entornos locales en programas de aplicación

Los entornos locales se implantan de una manera en los sistemas Windows® y de otra en los sistemas basados en UNIX®. Hay dos entornos locales en sistemas basados en UNIX:

- El entorno local del entorno le permite especificar el idioma, el símbolo de moneda, etc. que desea utilizar.
- El entorno local del programa contiene el idioma, símbolo de moneda, etc. actuales de un programa que se está ejecutando.

En sistemas Windows, las preferencias culturales se pueden definir mediante Configuración regional en el Panel de control. Sin embargo, no hay ningún entorno local del entorno como el de sistemas basados en UNIX.

Cuando se inicia el programa, obtiene un entorno local C por omisión. *No* obtiene una copia del entorno local del entorno. Si define como entorno local del programa cualquier entorno local que no sea "C", DB2 Universal Database utiliza el entorno local del programa actual para determinar los valores de página de códigos y de territorio correspondientes al entorno de la aplicación. Si no es así, estos valores se obtienen del entorno del sistema operativo. Tenga en cuenta que `setlocale()` no está libre de hebras y que si emite `setlocale()` desde dentro de la aplicación, el nuevo entorno local se establece para el proceso entero.

Cómo DB2 deriva entornos locales

En los sistemas basados en UNIX®, el entorno local activo utilizado por DB2® se determina a partir de la porción LC_CTYPE del valor de entorno local. Para ver detalles, consulte la documentación de NLS correspondiente a su sistema operativo.

- Si LC_CTYPE del entorno local del programa tiene un valor distinto de C, DB2 utilizará este valor para determinar la página de códigos de la aplicación, correlacionándolo con su página de códigos correspondiente.
- Si LC_CTYPE tiene el valor C (el entorno local C), DB2 definirá el entorno local del programa de acuerdo con el entorno local del entorno, mediante la función `setlocale()`.
- Si LC_CTYPE sigue teniendo el valor C, DB2 adoptará el valor por omisión de entorno Inglés de EE.UU. y la página de códigos 819 (ISO 8859-1).
- Si LC_CTYPE ya no tiene el valor C, se utilizará su nuevo valor para correlacionarlo con una página de códigos correspondiente.

Información relacionada:

- "Supported territory codes and code pages" en la publicación *Administration Guide: Planning*

Consideraciones sobre aplicaciones

Las secciones siguientes describen consideraciones que debe tener en cuenta al codificar una aplicación.

Consideraciones sobre el soporte de idiomas nacionales y sobre el desarrollo de aplicaciones

Las series de caracteres constantes en sentencias de SQL estático se convierten en el momento de la vinculación de la página de códigos de la aplicación a la página de códigos de la base de datos, y luego se utilizan en el momento de la ejecución en esta representación de página de códigos de la base de datos. Para evitar estas conversiones si no se desean, puede utilizar variables del lenguaje principal en lugar de constantes de series.

Si el programa contiene series de caracteres constantes, debe precompilar, vincular, compilar y ejecutar la aplicación utilizando la misma página de códigos. Para una base de datos Unicode, debe utilizar variables del lenguaje principal en lugar de utilizar constantes de series. La razón de esta recomendación es que las conversiones de datos que realiza el servidor se pueden producir en la fase de vinculación y en la de ejecución. Esto puede resultar un problema si se utilizan series de caracteres contantes dentro del programa. Estas series incorporadas se convierten en el momento de la vinculación según la página de códigos que está en vigor durante la fase de vinculación. Los caracteres ASCII de siete bits son comunes a todas las páginas de códigos soportadas por DB2 Universal Database y no ocasionan problemas. Para caracteres no ASCII, los usuarios se deben asegurar de que se utilizan las mismas tablas de conversión durante la vinculación y la ejecución con la misma página de códigos activa.

Se supone que cualquier dato externo que obtiene la aplicación está en la página de códigos de la aplicación. Esto incluye datos obtenidos de un archivo o de entrada del usuario. Asegúrese de que los datos procedentes de fuentes externas a la aplicación utilizan la misma página de códigos que la aplicación.

Si utiliza variables del lenguaje principal que utilizan datos gráficos en aplicaciones C o C++, hay temas especiales relacionados con el precompilador, el rendimiento de la aplicación y el diseño de la aplicación que debe tener en cuenta. Si utiliza juegos de códigos EUC en las aplicaciones, consulte los temas relacionados para ver las directrices generales.

Cuando desarrolle una aplicación, debe revisar los temas que siguen a este. Si no sigue las recomendaciones que se describen en estos temas, se pueden producir condiciones inesperadas. Estas condiciones las puede detectar el gestor de bases de datos, así que emitirá un mensaje de error o de aviso. Por ejemplo, una aplicación C contiene las siguientes sentencias de SQL que operan contra una tabla T1 con una columna definida como C1 CHAR(20):

```
     (0)  EXEC SQL CONNECT TO GLOBALDB;
     (1)  EXEC SQL INSERT INTO T1 VALUES ('a-constant');
          strcpy(sqlstmt, "SELECT C1 FROM T1 WHERE C1='a-constant');
     (2)  EXEC SQL PREPARE S1 FROM :sqlstmt;
Donde:
    página de códigos de la aplicación en el momento de la vinculación = x
    página de códigos de la aplicación en el momento de la ejecución = y
    página de códigos de la base de datos = z
```

En el momento de la vinculación '*a-constant*' en la sentencia (1) se convierte de la página de códigos **x** a la página de códigos **z**. Esta conversión se representa como (x→z).

En el momento de la ejecución, '*a-constant*' (x→z) se inserta en la tabla cuando se ejecuta la sentencia (1). Sin embargo, la cláusula WHERE de la sentencia (2) se ejecutará con '*a-constant*' (y→z). Si los puntos de código de la constante son tales que las dos conversiones (x→z e y→z) generan resultados diferentes, la operación SELECT de la sentencia (2) no podrá recuperar los datos insertados por la sentencia (1).

Conceptos relacionados:

- "Variables gráficas del lenguaje principal en C y C++" en la página 158
- "Derivación de valores de página de códigos" en la página 645
- "Consideraciones sobre los juegos de códigos EUC y UCS-2 de japonés y chino tradicional" en la página 656

Soporte de idiomas nacionales y sentencias de SQL

La codificación de sentencias de SQL no depende del idioma. Las palabras clave de SQL se deben escribir tal como se muestra, aunque se pueden escribir en mayúsculas, en minúsculas o en una combinación de ambas. Los caracteres de los nombres de objetos de base de datos, variables del lenguaje principal y etiquetas de programa que aparecen en una sentencia de SQL deben recibir soporte de la página de códigos de la aplicación.

El servidor no convierte nombres de archivo. Para codificar un nombre de archivo, puede utilizar el juego invariante ASCII o proporcionar la vía de acceso en los valores hexadecimales que están físicamente almacenados en el sistema de archivos.

En un entorno de varios bytes, hay cuatro caracteres que se consideran especiales que no pertenecen al juego de caracteres invariante. Estos caracteres son:

- Los caracteres de porcentaje de doble byte y de subrayado de doble byte que se utilizan en proceso LIKE.
- El carácter de espacio de doble byte que se utiliza, entre otras cosas, para rellenar con blancos series gráficas.
- El carácter de sustitución de doble byte, que se utiliza como sustitución durante la conversión de página de códigos cuando no existe ninguna correlación entre una página de códigos fuente y una página de códigos de destino.

Los puntos de código para cada uno de estos caracteres, por página de códigos, son los siguientes:

Tabla 85. Puntos de código para caracteres especiales de doble byte

Página de códigos	Porcentaje de doble byte	Subrayado de doble byte	Espacio de doble byte	Carácter de sustitución de doble byte
932	X'8193'	X'8151'	X'8140'	X'FCFC'
938	X'8193'	X'8151'	X'8140'	X'FCFC'
942	X'8193'	X'8151'	X'8140'	X'FCFC'
943	X'8193'	X'8151'	X'8140'	X'FCFC'
948	X'8193'	X'8151'	X'8140'	X'FCFC'
949	X'A3A5'	X'A3DF'	X'A1A1'	X'AFFE'
950	X'A248'	X'A1C4'	X'A140'	X'C8FE'
954	X'A1F3'	X'A1B2'	X'A1A1'	X'F4FE'
964	X'A2E8'	X'A2A5'	X'A1A1'	X'FDFE'
970	X'A3A5'	X'A3DF'	X'A1A1'	X'AFFE'
1381	X'A3A5'	X'A3DF'	X'A1A1'	X'FEFE'
1383	X'A3A5'	X'A3DF'	X'A1A1'	X'A1A1'
13488	X'FF05'	X'FF3F'	X'3000'	X'FFFD'
1363	X'A3A5'	X'A3DF'	X'A1A1'	X'A1E0'
1386	X'A3A5'	X'A3DF'	X'A1A1'	X'FEFE'
5039	X'8193'	X'8151'	X'8140'	X'FCFC'

Para bases de datos Unicode, el espacio GRAPHIC es X'0020', que es distinto del espacio GRAPHIC X'3000' que se utiliza para bases de datos euc-Japan y euc-Taiwan. Tanto X'0020' como X'3000' son caracteres de espacio en estándar Unicode. La diferencia en los puntos de código de espacio GRAPHIC se debe tener en cuenta cuando se comparan datos procedentes de estas bases de datos EUC con datos procedentes de una base de datos Unicode.

Información relacionada:

- "Predicado LIKE" en la publicación *Consulta de SQL, Volumen 1*
- "Juegos de caracteres de Extended UNIX Code (EUC)" en la página 654

Rutinas remotas

Cuando se codifican rutinas que se ejecutarán de forma remota, se deben tener en cuenta las consideraciones siguientes:

- Los datos de una rutina deben pertenecer a la página de códigos definida por la opción PARAMETER CCSID, que se especificó implícita o explícitamente al crear la rutina.
- La página de códigos de los datos de tipo carácter intercambiados con una rutina se convertirá a la página de códigos de la sección o de la rutina según convenga. Por tanto, los datos numéricos y estructuras de datos nunca se deben pasar con un tipo carácter si la página de códigos de la aplicación cliente es distinta de la página de códigos de la sentencia o rutina. Para evitar la conversión de caracteres, puede pasar datos definiéndolos en formato de serie binaria utilizando un tipo de datos BLOB o definiendo los datos de tipo carácter como FOR BIT DATA.

Por omisión, cuando invoca rutinas, se ejecutan bajo un entorno de idioma nacional por omisión, que puede no coincidir con el entorno de idioma nacional de la base de datos. Por lo tanto, la utilización de operaciones específicas de país/región o de página de códigos, como funciones y variables del lenguaje principal gráficas `wchar_t` de C, puede no funcionar según lo esperado. Asegúrese, si es necesario, de que se inicialice el entorno correcto cuando invoque la rutina.

Consideraciones sobre nombres de paquetes en entornos de páginas de códigos combinadas

Los nombres de paquetes se determinan cuando se invoca la API o mandato PRECOMPILE PROGRAM. Por omisión, se generan tomando los ocho primeros bytes del archivo fuente del programa de aplicación (sin la extensión de archivos) y se convierten a mayúsculas. También se puede definir un nombre de forma explícita. Independientemente del origen de un nombre de paquete, si trabaja en un entorno de páginas de códigos que no son iguales, los caracteres de los nombres de paquetes deben estar en un juego de caracteres invariante. De lo contrario pueden producirse problemas relacionados con la modificación del nombre de paquete. El gestor de bases de datos no podrá encontrar el paquete correspondiente a la aplicación o una herramienta del cliente no mostrará el nombre correcto del paquete.

Se producirá una modificación de nombre de paquete debida a conversión de caracteres si alguno de los caracteres del nombre del paquete no se correlaciona directamente con un carácter válido de la página de códigos de la base de datos. En estos casos, un carácter de sustitución sustituye al carácter que no se ha convertido. Tras dicha modificación, es posible que el nombre del paquete, cuando

se vuelve a convertir a la página de códigos de la aplicación, no coincida con el nombre original del paquete. Un ejemplo de un caso en que no se desea este comportamiento es cuando utiliza el Centro de control para listar paquetes y para trabajar con los mismos. Es posible que los nombres de paquetes que se muestran no coincidan con los nombres esperados.

Para evitar problemas de conversión con nombres de paquetes, asegúrese de que sólo se utilizan caracteres que son válidos para la página de códigos de la aplicación y para la de la base de datos.

Página de códigos activa para precompilación y vinculación

En el momento de la precompilación/vinculación, el precompilador es la aplicación que se ejecuta. Se utiliza la página de códigos activa cuando se realizó la conexión con la base de datos antes de la petición de precompilación para sentencias precompiladas y para los datos que se devuelven en la SQLCA.

Conceptos relacionados:

- "Página de códigos activa para la ejecución de aplicaciones" en la página 650

Página de códigos activa para la ejecución de aplicaciones

En el momento de la ejecución, la página de códigos activa de la aplicación de usuario cuando se realiza la conexión con la base de datos permanece en vigor mientras dura la conexión. Todos los datos se interpretan según esta página de códigos; esto incluye sentencias de SQL dinámico, datos de entrada del usuario, datos de salida del usuario y campos de caracteres de la SQLCA.

Conceptos relacionados:

- "Página de códigos activa para precompilación y vinculación" en la página 650

Conversión de caracteres entre páginas de códigos diferentes

Preferiblemente, para conseguir un rendimiento óptimo, las aplicaciones deberían utilizar siempre la misma página de códigos que las sentencias invocadas desde la aplicación. Sin embargo, esto no siempre es posible o resulta práctico. Los productos DB2® proporcionan soporte para la conversión de página de códigos que permite que la aplicación y la base de datos utilicen distintas páginas de códigos. Los caracteres de una página de códigos se deben correlacionar con la otra página de códigos para mantener la integridad de los datos.

Cuándo se produce conversión de la página de códigos

La conversión de página de códigos se puede producir en las siguientes situaciones:

- Cuando un cliente o aplicación que accede a una base de datos se ejecuta en una página de códigos distinta de la página de códigos de la sentencia invocada:

  En algunas situaciones puede minimizar o eliminar la conversión de caracteres cliente/servidor. Por ejemplo, puede:

  - Crear una base de datos en Windows NT que utilice la página de códigos 850 para que coincida con un entorno de aplicaciones cliente de Windows® que utiliza predominantemente la página de códigos 850.

Si se utiliza una aplicación ODBC de Windows con el controlador ODBC de DB2® de IBM® en el cliente de bases de datos de Windows, este problema se puede mitigar utilizando las palabras clave TRANSLATEDLL y TRANSLATEOPTION en el archivo `odbc.ini` o `db2cli.ini`.

  - Crear una base de datos en AIX® que utilice la página de códigos 850 para que coincida con un entorno de aplicaciones cliente que utiliza predominantemente la página de códigos 850.
  - Evite especificar la opción CCSID al crear tablas y la opción PARAMETER CCSID al crear rutinas.

- Cuando un cliente o aplicación que importa un archivo PC/IXF se ejecuta en una página de códigos distinta de la del archivo que se va a importar.

  Esta conversión de datos se producirá en la máquina cliente de bases de datos antes de que el cliente acceda al servidor de bases de datos. Es posible que se lleve a cabo una conversión de datos adicional si la aplicación se ejecuta en una página de códigos distinta de la de la base de datos (tal como se ha indicado en el punto anterior).

  La conversión de datos, si se realiza, también depende del modo en que se haya llamado al programa de utilidad de importación.

- Cuando se utiliza DB2 Connect para acceder a datos de un sistema principal, sistema AS/400® o servidor iSeries. En este caso, el receptor de los datos convierte los datos de caracteres. Por ejemplo, DB2 para MVS/ESA convierte al identificador de juego de caracteres codificados (CCSID) adecuado de MVS™ los datos que se envían a DB2 para MVS/ESA. DB2 Connect convierte los datos que se envían a la máquina DB2 Connect desde DB2 para MVS/ESA.

No se producirá conversión de caracteres para:

- Nombres de archivos. Debe utilizar el juego invariable ASCII para nombres de archivos o debe proporcionar el nombre de archivo en los valores hexadecimales que están físicamente almacenados en el sistema de archivos. Tenga en cuenta que, si incluye un nombre de archivo como parte de una sentencia de SQL, se convierte como parte de la conversión de la sentencia.
- Los datos destinados a una columna que tiene asignado el atributo FOR BIT DATA o que proceden de ésta o los datos que se utilizan en una operación de SQL cuyo resultado es datos FOR BIT o BLOB. En estos casos, los datos se tratan como una corriente de bytes y no se produce ninguna conversión.

  Nota: Un literal insertado en una columna definida como FOR BIT DATA se podría convertir si dicho literal formara parte de una sentencia de SQL que se ha convertido.

- Un producto o plataforma DB2 que no dé soporte, o que no tenga el soporte instalado, a la combinación deseada de páginas de códigos. En este caso, se devuelve un SQLCODE -332 (SQLSTATE 57017) si intenta ejecutar la aplicación.

Conceptos relacionados:

- "Conversión de caracteres" en la publicación *Consulta de SQL, Volumen 1*

Sustitución de caracteres durante conversiones de páginas de códigos

Si la aplicación realiza una conversión de una página de códigos a otra, es posible que uno o más caracteres no estén representados en la página de códigos de destino. Si esto sucede, DB2 inserta un carácter de *sustitución* en la serie de destino en lugar del carácter que no tiene representación. El carácter de sustitución se

considera una parte válida de la serie. En situaciones en las que se produce una sustitución, el indicador SQLWARN10 de la SQLCA adopta el valor 'W'.

Nota: Cualquier otra conversión de caracteres resultante del uso de la opción del precompilador WCHARTYPE CONVERT no marcarán un aviso si se realiza alguna sustitución.

Conceptos relacionados:

- "Opción del precompilador WCHARTYPE en C y C++" en la página 174

Información relacionada:

- "Mandato PRECOMPILE" en la publicación *Consulta de mandatos*

Conversiones de páginas de códigos soportadas

Cuando se produce una conversión de datos, la conversión tiene lugar de una *página de códigos fuente* a una *página de códigos de destino*.

La página de códigos fuente se determina a partir de la fuente de los datos; los datos procedentes de la aplicación tienen una página de códigos fuente igual a la página de códigos de la aplicación y los datos procedentes de la base de datos tienen una página de códigos fuente igual a la página de códigos de la base de datos.

La determinación de la página de códigos de destino es más complicada; hay que tener en cuenta dónde se van a colocar los datos, incluidas las normas correspondientes a operaciones intermedias:

- Si los datos se mueven directamente desde una aplicación a una base de datos, sin que intervengan operaciones, la página de códigos de destino es la de la base de datos.
- Si los datos se importan en una base de datos desde un archivo PC/IXF, hay dos pasos de conversión de caracteres:
  1. Desde la página de códigos del archivo PC/IXF (página de códigos fuente) a la página de códigos de la aplicación (página de códigos de destino)
  2. Desde la página de códigos de la aplicación (página de códigos fuente) a la página de códigos de la base de datos (página de códigos de destino)

  Debe tener cuidado en situaciones en las que se pueden producir dos pasos de conversión. Para evitar una posible pérdida de datos de caracteres, asegúrese de seguir las conversiones de caracteres soportadas. Además, dentro de cada grupo, solo los caracteres que existen en ambas páginas de códigos, fuente y destino, tienen conversiones significativas. Se utilizan otros caracteres como *sustituciones* y sólo resultan útiles para convertir desde la página de códigos de destino de nuevo a la página de códigos fuente (y no necesariamente proporcionan conversiones significativas en el proceso de conversión de dos pasos mencionado anteriormente). Estos problemas se pueden evitar si la página de códigos de la aplicación coincide con la de la base de datos.
- Si los datos se derivan de operaciones realizadas en datos de caracteres, en los que la fuente puede ser la página de códigos de la aplicación, la de la base de datos, FOR BIT DATA o datos BLOB, la conversión de datos se basa en un conjunto de normas. Es posible que algunos de los elementos de datos, o todos ellos, se tengan que convertir a un resultado intermedio antes de que se pueda determinar la página de códigos de destino final.

Nota: Las conversiones de páginas de códigos entre páginas de códigos de varios bytes, por ejemplo DBCS y EUC, pueden dar como resultado un aumento o reducción de la longitud de la serie.

Conceptos relacionados:

- "Conversión de caracteres" en la publicación *Consulta de SQL, Volumen 1*
- "Conversión de caracteres entre páginas de códigos diferentes" en la página 650

Información relacionada:

- "Supported territory codes and code pages" en la publicación *Administration Guide: Planning*

Factor de expansión de conversión de página de códigos

Si la aplicación completa satisfactoriamente un intento de establecer conexión con un servidor de bases de datos DB2, debe tener en cuenta los siguientes campos en la SQLCA que se devuelve:

- El segundo símbolo del campo SQLERRMC (los símbolos se separan mediante X'FF') indica la página de códigos de la base de datos. El noveno símbolo del campo SQLERRMC indica la página de códigos de la aplicación. Al consultar la página de códigos de la aplicación y compararla con la página de códigos de la base de datos se indica a la aplicación si ha establecido una conexión que experimentará conversiones de caracteres.
- La primera y segunda entrada de la matriz SQLERRD. SQLERRD(1) contiene un valor entero igual al factor máximo esperado de expansión o contracción correspondiente a la longitud de datos de caracteres mixtos (tipos de datos CHAR) cuando se convierten a la página de códigos de la base de datos desde la página de códigos de la aplicación. SQLERRD(2) contiene un valor entero igual al factor máximo esperado de expansión o contracción correspondiente a la longitud de datos de caracteres mixtos (tipos de datos CHAR) cuando se convierten a la página de códigos de la aplicación desde la página de códigos de la base de datos. Un valor 0 ó 1 indica que no hay expansión; un valor mayor que 1 indica una posible expansión de la longitud; un valor negativo indica una posible contracción.

Las consideraciones correspondientes a datos de series gráficas no deberían tenerse en cuenta en situaciones de páginas de códigos distintas. Cada serie tiene siempre el mismo número de caracteres, independientemente de si los datos están en la página de códigos de la aplicación o en la de la base de datos.

Conceptos relacionados:

- "Desarrollo de aplicaciones en situaciones de páginas de códigos distintas" en la página 660

Información relacionada:

- "Sentencia CONNECT (Tipo 1)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CONNECT (Tipo 2)" en la publicación *Consulta de SQL, Volumen 2*

Juegos de caracteres DBCS

Cada página de códigos combinada de juego de caracteres de un solo byte (SBCS) o de juego de caracteres de doble byte (DBCS) permite puntos de código de caracteres de un solo byte y de doble byte. Esto se suele conseguir reservando un subconjunto de los 256 puntos de códigos disponibles de una tabla de códigos mixta para caracteres de un solo byte, con el resto de puntos de código no definidos o asignados al primer byte de puntos de código de doble byte. Estos puntos de código se muestran en la tabla siguiente.

Tabla 86. Puntos de código de juegos de caracteres mixtos

País/Región	Página de códigos mixta soportada	Puntos de código para caracteres de un solo byte	Puntos de código para el primer byte de caracteres de doble byte
Japón	932, 943	X'00'-X'7F', X'A1'-X'DF'	X'81'-X'9F', X'E0'-X'FC'
Japón	942	X'00'-X'80', X'A0'-X'DF', X'FD'-X'FF'	X'81'-X'9F', X'E0'-X'FC'
Taiwán	938 (*)	X'00'-X'7E'	X'81'-X'FC'
Taiwán	948 (*)	X'00'-X'80', X'FD', X'FE'	X'81'-X'FC'
Corea	949	X'00'-X'7F'	X'8F'-X'FE'
Taiwán	950	X'00'-X'7E'	X'81'-X'FE'
China	1381	X'00'-X'7F'	X'8C'-X'FE'
Corea	1363	X'00'-X'7F'	X'81'-X'FE'
China	1386	X'00'	X'81'-X'FE'
Nota: (*) Esta es una página de códigos antigua que ya no se recomienda.			

Los puntos de código que no están asignados a ninguna de estas categorías no están definidos y se procesan como puntos de código no definidos de un solo byte.

Dentro de cada tabla de códigos DBCS implícita, hay 256 puntos de código disponibles como el segundo byte para cada primer byte válido. Los valores de segundo byte pueden adoptar cualquier valor comprendido entre X'40' y X'7E' y entre X'80' y X'FE'. Tenga en cuenta que, en entornos DBCS, DB2 no realiza la comprobación de validez de caracteres individuales de doble byte.

Juegos de caracteres de Extended UNIX Code (EUC)

Cada página de códigos EUC permite puntos de código de caracteres de un solo byte y un máximo de tres juegos diferentes de puntos de código de caracteres de varios bytes. Este soporte se consigue reservando un subconjunto de los 256 puntos de código disponibles de cada identificador de página de códigos SBCS relacionado para caracteres de un solo byte. El resto de los puntos de código son indefinidos, están asignados como un elemento de un carácter de varios bytes o están asignados como un introductor de un solo desplazamiento de un carácter de varios bytes. Estos puntos de código se muestran en las tablas siguientes.

Tabla 87. Puntos de código EUC japonés

Grupo	1er byte	2º byte	3er byte	4º byte
G0	X'20'-X'7E'	n/d	n/d	n/d
G1	X'A1'-X'FE'	X'A1'-X'FE'	n/d	n/d
G2	X'8E'	X'A1'-X'FE'	n/d	n/d
G3	X'8E'	X'A1'-X'FE'	X'A1'-X'FE'	n/d

Tabla 88. Puntos de código EUC coreano

Grupo	1er byte	2º byte	3er byte	4º byte
G0	X'20'-X'7E'	n/d	n/d	n/d
G1	X'A1'-X'FE'	X'A1'-X'FE'	n/d	n/d
G2	n/d	n/d	n/d	n/d
G3	n/d	n/d	n/d	n/d

Tabla 89. Puntos de código EUC chino tradicional

Grupo	1er byte	2º byte	3er byte	4º byte
G0	X'20'-X'7E'	n/d	n/d	n/d
G1	X'A1'-X'FE'	X'A1'-X'FE'	n/d	n/d
G2	X'8E'	X'A1'-X'FE'	X'A1'-X'FE'	X'A1'-X'FE'
G3	n/d	n/d	n/d	n/d

Tabla 90. Puntos de código EUC chino simplificado

Grupo	1er byte	2º byte	3er byte	4º byte
G0	X'20'-X'7E'	n/d	n/d	n/d
G1	X'A1'-X'FE'	X'A1'-X'FE'	n/d	n/d
G2	n/d	n/d	n/d	n/d
G3	n/d	n/d	n/d	n/d

Los puntos de código que no están asignados a ninguna de estas categorías no están definidos y se procesan como puntos de código no definidos de un solo byte.

Programas CLI, ODBC, JDBC y SQLJ en un entorno DBCS

Los programas JDBC y SQLJ acceden a DB2® mediante el controlador CLI/ODBC de DB2 y por lo tanto utilizan el mismo archivo de configuración (db2cli.ini). Se deben añadir las siguientes entradas a este archivo de configuración si se ejecutan programas Java™ que acceden a DB2 Universal Database en un entorno DBCS:

PATCH1 = 65536
Impone que el controlador inserte manualmente una ″G″ delante de los literales de caracteres que de hecho sean literales gráficos. Este valor de PATCH1 se debe establecer siempre que se trabaje en un entorno de doble byte.

PATCH1 = 64
Impone que el controlador termine en NULL las series de salida gráficas. Es necesario para Microsoft® Access en un entorno de doble byte. Si

también tiene que utilizar este valor de PATCH1, deberá añadir los dos valores juntos (64+65536 = 65600) y establecer PATCH1=65600. Consulte la nota 2 siguiente para obtener más información sobre cómo especificar valores de PATCH1.

PATCH2 = 7
Impone que el controlador correlacione todos los tipos de datos de columnas de gráficos con el tipo de datos de columna de caracteres. Este valor de PATCH2 es necesario en un entorno de doble byte.

PATCH2 = 10
Sólo se debe utilizar en un entorno EUC (Extended Unix Code). Este valor de PATCH2 asegura que el controlador CLI proporciona datos correspondientes a variables de caracteres (CHAR, VARCHAR, etc.) en el formato adecuado para el controlador JDBC. Los datos de estos tipos de caracteres no se podrán utilizar en JDBC sin este valor.

Notas:

1. Cada una de estas palabras clave se establece en cada sección específica de base de datos del archivo db2cli.ini. Si desea establecerlas para varias bases de datos, repítalas para cada sección de base de datos de db2cli.ini.
2. Para establecer varios valores de PATCH1, añada los valores individuales y utilice la suma. Para establecer PATCH1 en 64 y 65536, establezca PATCH1=65600 (64+65536). Si ya tiene establecidos otros valores de PATCH1, sustituya el número existente por la suma de dicho número y los nuevos valores de PATCH1 que desee añadir.
3. Para establecer varios valores de PATCH2, especifíquelos en una serie delimitada por comas (a diferencia de la opción PATCH1). Para establecer los valores 1 y 7 para PATCH2, establezca PATCH2=″1,7″

Consideraciones sobre los juegos de códigos EUC y UCS-2 de japonés y chino tradicional

Las secciones siguientes describen las consideraciones a tener en cuenta para los juegos de códigos EUC y UCS-2 de japonés y chino tradicional.

Consideraciones sobre los juegos de códigos EUC y UCS-2 de japonés y chino tradicional

Extended UNIX® Code (EUC) indica un conjunto de normas generales de codificación que pueden soportar entre uno y cuatro juegos de caracteres en los entornos operativos basados en UNIX. Las normas de codificación se basan en la definición ISO 2022 para la codificación de datos de 7 bits y de 8 bits, en los que se utilizan caracteres de control para separar algunos de los juegos de caracteres. Un juego de códigos basado en EUC se ajusta a las normas de codificación de EUC, pero también identifica los juegos de caracteres específicos asociados a las instancias concretas. Por ejemplo, el juego de códigos IBM®-eucJP para el japonés remite a la codificación de caracteres JIS (Japanese Industrial Standard) según las normas de codificación de EUC.

El soporte de datos gráficos (caracteres puros de doble byte) por parte de la base de datos y de la aplicación cliente, mientras se ejecutan bajo páginas de códigos EUC con una codificación de caracteres de longitud superior a dos bytes, es limitado. Los productos DB2 Universal Database implementan normas estrictas para los datos gráficos que requieren que todos los caracteres contengan exactamente dos bytes. Estas normas no admiten muchos caracteres de las páginas

de códigos EUC del japonés ni del chino tradicional. Para vencer esta situación, se proporciona soporte, tanto a nivel de aplicación como a nivel de base de datos, para representar los datos gráficos EUC en japonés y en chino tradicional utilizando otro esquema de codificación.

Una base de datos creada bajo EUC de japonés o de chino tradicional realmente almacenará y manipulará los datos gráficos utilizando el juego de códigos Unicode UCS-2, esquema de codificación de doble byte que constituye un subconjunto adecuado del repertorio completo de caracteres Unicode. De forma parecida, una aplicación que se ejecute bajo estas páginas de códigos enviará los datos gráficos a la base de datos en forma de datos codificados en UCS-2. Mediante este soporte, las aplicaciones que se ejecutan bajo páginas de códigos EUC pueden acceder a los mismos tipos de datos que las que se ejecutan bajo páginas de códigos DBCS. El identificador de página de códigos definido por IBM asociado a UCS-2 es 1200, y el número de CCSID para la misma página de códigos es 13488. Los datos gráficos de una base de datos eucJP o eucTW utilizan el número de CCSID 13488. En una base de datos Unicode, utilice el CCSID 1200 para los datos GRAPHIC (gráficos).

DB2 Universal Database soporta todos los caracteres Unicode que se pueden codificar utilizando UCS-2, pero no realiza ninguna composición, descomposición ni normalización de los caracteres. Puede encontrar más información acerca del estándar Unicode en el sitio de Unicode Consortium, `www.unicode.org`, y en la última edición de la publicación sobre el estándar Unicode publicada por Addison Wesley Longman, Inc.

Si trabaja con aplicaciones o bases de datos que utilizan estos juegos de caracteres, es posible que tenga que considerar la posibilidad de utilizar datos codificados UCS-2. Cuando se convierten datos gráficos UCS-2 a la página de códigos EUC de la aplicación, existe la posibilidad de que se incremente la longitud de los datos. Si se visualizan grandes cantidades de datos, puede que sea necesario asignar almacenamientos intermedios, convertir y visualizar los datos en una serie de fragmentos.

En los apartados siguientes se explica cómo manejar los datos en este entorno. En estas secciones, el término EUC se utiliza únicamente para hacer referencia a los juegos de caracteres EUC del japonés y del chino tradicional. Observe que las explicaciones no se aplican al soporte por parte de DB2 de EUC del coreano y del chino simplificado, puesto que los datos gráficos de estos juegos de caracteres se representan utilizando la codificación EUC.

Conceptos relacionados:

- "Factor de expansión de conversión de página de códigos" en la página 653
- "Desbordamiento de longitud de serie de conversión de página de códigos en entornos de juegos de códigos mixtos" en la página 666

Información relacionada:

- "Supported territory codes and code pages" en la publicación *Administration Guide: Planning*
- "Juegos de caracteres de Extended UNIX Code (EUC)" en la página 654

Consideraciones sobre las bases de datos y los clientes de doble byte con EUC mixtos

La administración de objetos de base de datos en entornos con páginas mixtas de códigos EUC y de doble byte resulta complicada por la posible expansión o contracción de la longitud de los nombres de objeto como consecuencia de conversiones entre la página de códigos del cliente y la de la base de datos. En concreto, muchos mandatos de administración y programas de utilidad tienen documentados límites en la longitud de las series de caracteres que pueden tomar como parámetros de entrada o salida. Estos límites se suelen imponer en el cliente, a no ser que se documente lo contrario. Por ejemplo, el límite para un nombre de tabla es de 128 bytes. Es posible que una serie de caracteres que contiene 128 bytes bajo una página de códigos de doble byte sea mayor, pongamos de 135 bytes, bajo una página de códigos EUC. Los mandatos tales como REORGANIZE TABLE no considerarían válido este hipotético nombre de tabla de 135 bytes si se utiliza como parámetro de entrada, a pesar de que sea válido en la base de datos de doble byte de destino. De forma parecida, la longitud máxima permitida para los parámetros de salida se puede exceder, después de una conversión de la página de códigos de la base de datos a la página de códigos de la aplicación. Esto puede ocasionar un error de conversión o que se produzca un truncamiento de los datos de salida.

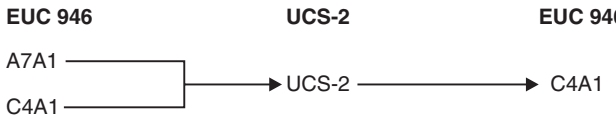
Si prevé que va a hacer un amplio uso de mandatos de administración y programas de utilidad en un entorno mixto con EUC y doble byte, debe definir los objetos de base de datos y los datos asociados a los mismos con la posibilidad de que la longitud supere los límites soportados. La administración de una base de datos EUC desde un cliente de doble byte impone menos restricciones que la administración de una base de datos de doble byte desde un cliente EUC. Normalmente, las series de caracteres de doble byte son iguales o más cortas que las series de caracteres EUC correspondientes. Esta característica generalmente conducirá a que se produzcan menos problemas debidos a la imposición de los límites en la longitud de las series de caracteres.

Nota: En el caso de las sentencias de SQL, la validación de los parámetros de entrada no se lleva a cabo hasta que se ha convertido la sentencia completa a la página de códigos de la base de datos. Así, se pueden utilizar series de caracteres que pueden ser técnicamente más largas de lo permitido cuando se representen en la página de códigos del cliente, pero que se ajusten a los requisitos de longitud cuando se representen en la página de códigos de la base de datos.

Consideraciones sobre la conversión de caracteres para usuarios de chino tradicional

Debido a la definición de estándares para el chino tradicional, existe un efecto colateral con que se puede encontrar cuando convierta algunos caracteres de páginas de códigos EUC o de doble byte y UCS-2. Existen 189 caracteres (que consisten en 187 radicales y 2 números) que comparten el mismo elemento de código UCS-2, cuando se convierten, que otro carácter del juego de códigos. Cuando estos caracteres se vuelven a convertir a EUC o doble byte, se convierten al elemento de código del mismo ideograma UCS-2, en lugar del elemento de código original. Cuando se visualizan, los caracteres parecen iguales, pero tienen un elemento de código distinto. Según el diseño de la aplicación, es posible que deba tener en cuenta este comportamiento.

Como ejemplo, considere lo que sucede con el elemento de código A7A1 en la página de códigos EUC 964, cuando se convierte a UCS-2 y luego se vuelve a convertir a la página de códigos original, EUC 946:



Así, los elementos de código originales A7A1 y C4A1 terminan siendo C4A1 después de la conversión.

Datos gráficos en aplicaciones EUC en japonés o chino tradicional

La información siguiente describe consideraciones sobre el desarrollo de aplicaciones EUC para datos gráficos, que incluyen constantes gráficas, datos gráficos en UDF, procedimientos almacenados, archivos DBCLOB y clasificación:

- Constantes gráficas

  Las constantes gráficas, o literales, se clasifican realmente como datos mixtos de tipo carácter, puesto que forman parte de una sentencia de SQL. El servidor de bases de datos convierte implícitamente a la codificación de gráficos cualquier constante gráfica de una sentencia de SQL procedente de un cliente EUC japonés o chino tradicional. En las aplicaciones SQL se pueden utilizar caracteres gráficos compuestos por caracteres codificados en EUC. Un servidor de bases de datos EUC convertirá dichos literales al juego de códigos de la base de datos gráficos, que será UCS-2. Las constantes gráficas procedentes de los clientes EUC nunca deben contener caracteres de una sola anchura, como por ejemplo los caracteres ASCII CS0 de 7 bits o los caracteres en japonés EUC CS2 (Katakana).
- UDF

  Se invoca a las UDF en el servidor de bases de datos y están destinadas a tratar con datos codificados en el mismo juego de códigos que la base de datos. En el caso de las bases de datos que se ejecutan bajo los juegos de caracteres del japonés o del chino tradicional, los datos mixtos de tipo carácter se codifican utilizando el juego de códigos EUC bajo el que se crea la base de datos. Los datos gráficos se codifican utilizando UCS-2. Las UDF tienen que reconocer y manejar datos gráficos codificados con UCS-2.

  Por ejemplo, supongamos que crea una UDF denominada VARCHAR y la UDF convierte una serie gráfica en una serie de caracteres mixtos. Si la base de datos se crea bajo el juego de códigos EUC, la función VARCHAR tendrá que convertir una serie gráfica codificada como UCS-2 en una representación EUC.
- Procedimientos almacenados

  Un procedimiento almacenado que se ejecute bajo un juego de códigos EUC del japonés o del chino tradicional debe ser capaz de reconocer y manejar datos gráficos codificados utilizando UCS-2. Con estos juegos de códigos, los datos gráficos recibidos o devueltos a través de la SQLDA de entrada/salida del procedimiento almacenado se codifican utilizando UCS-2.
- Archivos DBCLOB

  Las consideraciones importantes que deben tenerse en cuenta para archivos DBCLOB son:

  – Se supone que un archivo de datos DBCLOB está en la página de códigos EUC de la aplicación. Para los archivos DBCLOB en EUC, los datos se

convierten a UCS-2 en el cliente cuando se produce una lectura, y de UCS-2 en el cliente cuando se trata de una grabación.
  - El número de bytes leídos o grabados en el servidor se devuelve en el campo de longitud de datos de la variable de referencia a archivo. El número de bytes se basa en el número de caracteres codificados en UCS-2 que se leen del archivo o que se graban en el mismo. El número de bytes que realmente se lee del archivo o que se graba en el mismo puede ser mayor que lo que el servidor graba en el campo de longitud de datos.
- Clasificación

  Los datos gráficos se clasifican en secuencia binaria. Los datos mixtos se clasifican en la secuencia de clasificación de la base de datos aplicada en cada byte. Debido a las posibles diferencias en la clasificación de caracteres en un juego de códigos EUC y en un juego de códigos DBCS para el mismo país/región, se pueden obtener resultados distintos cuando se clasifican los mismos datos en una base de datos EUC y en una base de datos DBCS.

Información relacionada:

- "Función escalar GRAPHIC" en la publicación *Consulta de SQL, Volumen 1*
- "Sentencia SELECT" en la publicación *Consulta de SQL, Volumen 2*
- "Series gráficas" en la publicación *Consulta de SQL, Volumen 1*

Desarrollo de aplicaciones en situaciones de páginas de códigos distintas

En función de los esquemas de codificación de caracteres utilizados por la página de códigos de la aplicación y la página de códigos de la base de datos, se puede producir o no producir un cambio en la longitud de una serie cuando ésta se convierte de la página de códigos de origen a la página de códigos de destino. Un cambio en la longitud se suele asociar a conversiones entre páginas de códigos de múltiples bytes con distintos esquemas de codificación, por ejemplo DBCS y EUC.

Generalmente, un posible incremento de la longitud es más grave que una posible disminución de ésta, puesto que una sobreasignación de memoria resulta menos problemática que una subasignación. Hay que tratar por separado las consideraciones sobre las aplicaciones para enviar o recuperar datos en función del lugar en que se puede producir la posible expansión. También es importante observar las diferencias entre una situación de *mejor caso* y de *peor caso* cuando se indica una expansión o contracción de la longitud. Los valores positivos, que indican una posible expansión, darán el factor de multiplicación del *peor caso*. Por ejemplo, un valor de `2` para el campo SQLERRD(1) o SQLERRD(2) significa que se necesitará un almacenamiento máximo del doble de la longitud de la serie para manejar los datos después de la conversión. Éste es un indicador del *peor caso*. En este ejemplo, el *mejor caso* sería que, tras la conversión, la longitud no variara.

Los valores negativos para SQLERRD(1) o SQLERRD(2), que indican una posible contracción, también proporcionan el factor de expansión del *peor caso*. Por ejemplo, un valor de `-1` significa que el almacenamiento máximo necesario es igual a la longitud de la serie antes de la conversión. Claro que es posible que se necesite menos almacenamiento, pero prácticamente esto tiene poca utilidad, a no ser que la aplicación receptora sepa por adelantado cómo están estructurados los datos de origen.

Para asegurarse de que siempre tendrá asignado suficiente almacenamiento para abarcar la máxima expansión posible después de una conversión de caracteres, debe asignar un almacenamiento igual al valor de max_target_length obtenido del cálculo siguiente:

1. Determine el factor de expansión de los datos.

   Para las transferencias de datos desde la aplicación a la base de datos:

   ```
   expansion_factor = ABS[SQLERRD(1)]
   if expansion_factor = 0
      expansion_factor = 1
   ```

   Para las transferencias de datos desde la base de datos a la aplicación:

   ```
   expansion_factor = ABS[SQLERRD(2)]
   if expansion_factor = 0
      expansion_factor = 1
   ```

   En los cálculos anteriores, ABS hace referencia al valor absoluto.

   La comprobación de expansion_factor = 0 es necesaria, puesto que algunos productos DB2 Universal Database devuelven 0 en SQLERRD(1) y en SQLERRD(2). Estos servidores no soportan conversiones de página de códigos que tengan como resultado una expansión o una merma de los datos; esto se representa mediante un factor de expansión de 1.

2. Cálculo intermedio de la longitud.

   ```
   temp_target_length = actual_source_length * expansion_factor
   ```

3. Determine la longitud máxima para el tipo de datos de destino.

Tipo de datos de destino	Longitud máxima del tipo (type_maximum_lenght)
CHAR	254
VARCHAR	32.672
LONG VARCHAR	32.700
CLOB	2.147.483.647

4. Determine la longitud máxima de destino.

   ```
   1  if temp_target_length < actual_source_length
         max_target_length = type_maximum_length
      else
   2  if temp_target_length > type_maximum_length
         max_target_length = type_maximum_length
      else
   3  max_target_length = temp_target_length
   ```

   Todas las comprobaciones anteriores son necesarias para tener en cuenta los desbordamientos que se puedan producir durante el cálculo de la longitud. Las comprobaciones específicas son:

   1 Se produce un desbordamiento numérico durante el cálculo de temp_target_length en el paso 2.

   Si el resultado de multiplicar dos valores positivos juntos es mayor que el valor máximo para el tipo de datos, el resultado se *reinicia* y se devuelve un valor menor que el mayor de los dos.

   Por ejemplo, el valor máximo de un entero de 2 bytes con signo (que se utiliza para la longitud de los tipos de datos que no son CLOB) es 32.767. Si actual_source_length es 25.000 y el factor de expansión es 2, temp_target_lenght es, teóricamente, 50.000. Este valor es demasiado alto para el entero de 2 bytes con signo, por lo que se reinicia y se devuelve -15.536.

Para el tipo de datos CLOB, se utiliza para la longitud un entero de 4 bytes con signo. El valor máximo de un entero de 4 bytes con signo es 2.147.483.647.

2 `temp_target_length` es demasiado grande para el tipo de datos.

La longitud de un tipo de datos no puede superar los valores listados en el paso 3.

Si la conversión requiere más espacio del que hay disponible en el tipo de datos, puede existir la posibilidad de utilizar un tipo de datos más grande para que contenga el resultado. Por ejemplo, si un valor CHAR(250) requiere 500 bytes para contener la serie convertida, ésta no cabrá en un valor CHAR porque la longitud máxima es de 254 bytes. Sin embargo, existe la posibilidad de utilizar VARCHAR(500) para contener el resultado después de la conversión. Consulte el tema sobre desbordamiento de longitud de serie en conversión de página de códigos en entornos de juegos de códigos mixtos para obtener más información sobre lo que sucede cuando los datos convertidos sobrepasan el límite correspondiente a un tipo de datos.

3 `temp_target_lenght` es la longitud correcta para el resultado.

Utilizando los valores de SQLERRD(1) y SQLERRD(2) devueltos al conectar con la base de datos y los cálculos anteriores, puede determinar si existen posibilidades de que la longitud de una serie aumente o disminuya como consecuencia de la conversión de caracteres. En general, un valor de 0 ó 1 indica que no se producirá ninguna expansión; un valor superior a 1 indica una posible expansión de la longitud; un valor negativo indica una posible contracción. Observe que los valores de '0' sólo provendrán de productos DB2 Universal Database de versiones anteriores. Asimismo, estos valores no están definidos para otros productos de servidor de bases de datos. La tabla siguiente contiene los valores a esperar para diversas combinaciones de página de códigos de aplicación y página de códigos de base de datos cuando se utiliza DB2 Universal Database.

Tabla 91. Valores de SQLCA.SQLERRD en CONNECT

Página de códigos de la aplicación	Página de códigos de la base de datos	SQLERRD(1)	SQLERRD(2)
SBCS	SBCS	+1	+1
DBCS	DBCS	+1	+1
eucJP	eucJP	+1	+1
eucJP	DBCS	-1	+2
DBCS	eucJP	+2	-1
eucTW	eucTW	+1	+1
eucTW	DBCS	-1	+2
DBCS	eucTW	+2	-1
eucKR	eucKR	+1	+1
eucKR	DBCS	+1	+1
DBCS	eucKR	+1	+1
eucCN	eucCN	+1	+1
eucCN	DBCS	+1	+1
DBCS	eucCN	+1	+1

Si los valores SQLERRD(1) o SQLERRD(2) indican una expansión en el servidor de bases de datos o en el cliente de aplicación, debe tener en cuenta lo siguiente:

- Expansión en el servidor de bases de datos

  Si la entrada SQLERRD(1) indica una expansión en el servidor de bases de datos, la aplicación debe contemplar la posibilidad de que los datos de tipo carácter dependientes de la longitud que sean válidos en el cliente no lo sean en el servidor de bases de datos, después de que se hayan convertido. Por ejemplo, los productos DB2 requieren que los nombres de columna no tengan una longitud superior a 128 bytes. Es posible que una serie de caracteres que tenga una longitud de 128 bytes codificada bajo una página de códigos DBCS se expanda sobrepasando el límite de 128 bytes cuando se convierta a una página de códigos EUC. Esta posibilidad implica que pueden existir actividades que sean válidas cuando la página de códigos de la aplicación y la página de códigos de la base de datos sean iguales y no lo sean cuando sean distintas. Cuando diseñe bases de datos EUC y DBCS para situaciones de páginas de códigos distintas, proceda con precaución.
- Expansión en la aplicación

  Si la entrada SQLERRD(1) indica una expansión en la aplicación cliente, la aplicación debe contemplar la posibilidad de que los datos de tipo carácter dependientes de la longitud verán expandida su longitud una vez que se hayan convertido. Por ejemplo, se recupera una fila con una columna CHAR(128). Si las páginas de códigos de la base de datos y de la aplicación son iguales, la longitud de los datos devueltos es de 128 bytes. Sin embargo, en una situación de páginas de códigos distintos, 128 bytes de datos codificados bajo una página de códigos DBCS se pueden expandir hasta sobrepasar 128 bytes cuando se conviertan a una página de códigos EUC. Así, es posible que haya que asignar almacenamiento adicional para recuperar la serie completa.

Conceptos relacionados:

- "Desbordamiento de longitud de serie de conversión de página de códigos en entornos de juegos de códigos mixtos" en la página 666

Validación de parámetros basada en cliente en un entorno de juegos de códigos mixtos

Un efecto colateral importante de una potencial expansión o contracción de los datos de tipo carácter entre el cliente y el servidor implica la validación de los datos que se pasen entre la aplicación cliente y el servidor de bases de datos. En una situación de páginas de códigos desiguales, es posible que los datos que se determine que son válidos en el cliente sean realmente no válidos en el servidor de bases de datos después de una conversión de páginas de códigos. Y, a la inversa, los datos que no son válidos en el cliente pueden serlo en el servidor de bases de datos tras la conversión.

Existe la posibilidad de que cualquier aplicación de usuario final o biblioteca de API no sea capaz de manejar todas las posibilidades en una situación de páginas de códigos desiguales. Además, mientras que algunas validaciones de parámetros, como por ejemplo la longitud de las series, se realizan en el cliente para los mandatos y las API, las señales internas de las sentencias de SQL no se verifican hasta que se han convertido a la página de códigos de la base de datos. Esta verificación puede conducir a situaciones en que sea posible utilizar una sentencia de SQL en un entorno de páginas de códigos desiguales para acceder a un objeto de base de datos, como por ejemplo una tabla, pero que no sea posible acceder al mismo objeto utilizando una API o un mandato determinados.

Piense en una aplicación que devuelva datos contenidos en una tabla proporcionada por un usuario final y compruebe que el nombre de tabla no tiene una longitud superior a 128 bytes. Considere ahora los escenarios siguientes para esta aplicación:

1. Se crea una base de datos DBCS. Desde un cliente DBCS, se crea una tabla (t1) con un nombre de tabla que tiene una longitud de 128 bytes. El nombre de tabla incluye varios caracteres que constarían de más de dos bytes si se convirtiera la serie a EUC, lo que daría como resultado que la representación EUC del nombre de tabla tuviera una longitud total de 131 bytes. Puesto que no existe ninguna expansión para las conexiones de DBCS a DBCS, el nombre de tabla tendría 128 bytes en el entorno de la base de datos y la sentencia CREATE TABLE sería satisfactoria.
2. Un cliente EUC se conecta a la base de datos DBCS. Crea una tabla (t2) cuyo nombre de tabla tiene una longitud de 120 bytes cuando se codifica como EUC y de 100 bytes cuando se convierte a DBCS. El nombre de tabla en la base de datos DBCS tiene 100 bytes. La sentencia CREATE TABLE es satisfactoria.
3. El cliente EUC crea una tabla (t3) cuyo nombre de tabla contiene 64 caracteres EUC (131 bytes). Cuando se convierte este nombre a DBCS, su longitud se reduce hasta el límite de 128 bytes. La sentencia CREATE TABLE es satisfactoria.
4. El cliente EUC invoca a la aplicación para cada una de las tablas (t1, t2 y t3) de la base de datos DBCS, que da el resultado siguiente:

Tabla	Resultado
t1	La aplicación considera que el nombre de tabla no es válido porque tiene una longitud de 131 bytes.
t2	Visualiza resultados correctos.
t3	La aplicación considera que el nombre de tabla no es válido porque tiene una longitud de 131 bytes.

5. Se utiliza el cliente EUC para consultar la base de datos DBCS desde el CLP. Aunque el nombre de tabla tenga una longitud de 131 bytes en el cliente, las consultas resultan satisfactorias porque el nombre de tabla tiene una longitud de 128 bytes en el servidor.

Sentencia DESCRIBE en entornos de juegos de códigos mixtos

Una sentencia DESCRIBE realizada para una base de datos EUC devolverá información sobre columnas mixtas de caracteres y gráficos (GRAPHIC) en base a la definición de dichas columnas en la base de datos. Esta información se basa en la página de códigos del servidor antes de que se convierta a la página de códigos del cliente.

Cuando se ejecuta una sentencia DESCRIBE para un elemento de una lista de selección que se resuelve en el contexto de la aplicación (por ejemplo, `VALUES SUBSTR(?,1,2)`) para cualquier dato de tipo carácter o gráfico implicado, se debe evaluar el valor de SQLLEN devuelto, junto con la página de códigos devuelta. Si la página de códigos devuelta es la misma que la página de códigos de la aplicación, no se produce ninguna expansión. Si la página de códigos devuelta es la misma que la página de códigos de la base de datos, es posible que se produzca una expansión. Los elementos de lista de selección que son FOR BIT DATA (página

de códigos 0), o si en la página de códigos de la aplicación no se convierten al devolverlos a la aplicación, no se produce ninguna expansión ni contracción de la longitud informada.

Las consideraciones son distintas para una aplicación EUC que accede a una base de datos DBCS, en comparación con una aplicación DBCS que accede a una base de datos EUC:

- Aplicación EUC que accede a una base de datos DBCS

  Si la página de códigos de la aplicación es una página de códigos EUC, y si ésta emite una sentencia DESCRIBE para una base de datos que tiene una página de códigos DBCS, la información devuelta para las columnas CHAR y GRAPHIC se devuelve en el contexto de la base de datos. Por ejemplo, una columna CHAR(5) devuelta como parte de una sentencia DESCRIBE que tiene un valor de cinco en el campo SQLLEN. En el caso de datos que no son EUC, se asignan cinco bytes de almacenamiento cuando se captan los datos de esta columna. Con los datos EUC, el caso puede ser distinto. Si tiene lugar una conversión de página de códigos de DBCS a EUC, se puede producir un incremento de la longitud de los datos debido a la codificación distinta utilizada para los caracteres de las columnas CHAR. Por ejemplo, con el juego de caracteres del chino tradicional, el aumento máximo es el doble. Es decir, la longitud máxima de un carácter en la codificación DBCS es de dos bytes, que puede aumentar hasta una longitud máxima de cuatro bytes en EUC. Para el juego de códigos del japonés, el incremento máximo también es el doble. Sin embargo, observe que, mientras que la longitud máxima de un carácter en DBCS del japonés es de dos bytes, puede aumentar hasta un máximo de tres bytes en EUC del japonés. Aunque este incremento sólo parece corresponder a un factor de 1,5, los caracteres Katakana de un solo byte en DBCS del japonés tienen sólo una longitud de un byte, mientras que en el EUC del japonés tienen una longitud de dos bytes.

  Los cambios posibles en la longitud de los datos como consecuencia de conversiones de caracteres sólo se aplican a los datos mixtos de tipo carácter. La codificación de datos de caracteres gráficos siempre tiene la misma longitud, dos bytes, independientemente del esquema de codificación. Para evitar una pérdida de datos, es necesario evaluar si existe una situación de páginas de códigos desiguales, y si ésta se produce entre una aplicación EUC y una base de datos DBCS. Puede determinar la página de códigos de la base de datos y la de la aplicación mediante las señales de la SQLCA devuelta por una sentencia CONNECT. Si existe una situación de este tipo, es necesario que la aplicación asigne almacenamiento adicional para los datos mixtos de tipo carácter en base al factor de expansión máxima para ese esquema de codificación.
- Aplicación DBCS que accede a una base de datos EUC

  Si la página de códigos de la aplicación es una página de códigos DBCS y emite una sentencia DESCRIBE para una base de datos EUC, la situación es parecida a la que se produce cuando una aplicación EUC accede a una base de datos DBCS. Sin embargo, en esta situación es posible que la aplicación necesite menos almacenamiento que el indicado por el valor del campo SQLLEN. El peor caso en esta situación es que todos los datos sean de un solo byte o de doble byte bajo EUC, lo cual significa que se requieren exactamente SQLLEN bytes en el esquema de codificación DBCS. En cualquier otra situación se necesitan menos de SQLLEN bytes, puesto que se requiere un máximo de dos bytes para almacenar cualquier carácter EUC.

Conceptos relacionados:

- “Derivación de valores de página de códigos” en la página 645
- “Factor de expansión de conversión de página de códigos” en la página 653

- "Desbordamiento de longitud de serie de conversión de página de códigos en entornos de juegos de códigos mixtos" en la página 666

Información relacionada:

- "Sentencia DESCRIBE" en la publicación *Consulta de SQL, Volumen 2*

Datos de longitud fija y de longitud variable en entornos de juegos de códigos mixtos

Debido a un posible cambio en la longitud de las series cuando se producen conversiones entre páginas de códigos DBCS y EUC, debe contemplar la posibilidad de no utilizar tipos de datos de longitud fija. Según si necesita o no que se realice espacios en blanco de relleno, debe considerar la posibilidad de cambiar el SQLTYPE de una serie de caracteres de longitud fija por una serie de caracteres de longitud variable después de ejecutar la sentencia DESCRIBE. Por ejemplo, si se informa de una conexión de EUC a DBCS de un factor máximo de expansión de dos para una columna CHAR(5), la aplicación debe asignar diez bytes.

Si el SQLTYPE es de longitud fija, la aplicación EUC recibirá la columna como una corriente de datos EUC convertida a partir de los datos DBCS (que, en sí misma, puede tener hasta cinco bytes de espacios en blanco de relleno) con un espacio en blanco de relleno adicional si la conversión de página de códigos no ocasiona que el elemento de datos crezca hasta su tamaño máximo. Si el SQLTYPE es de longitud variable, se mantiene el significado original del contenido de la columna CHAR(5); no obstante, los cinco bytes de origen pueden tener un destino de entre cinco y diez bytes. De forma parecida, en caso de una posible reducción de los datos (aplicación DBCS y base de datos EUC), debe considerar la posibilidad de trabajar con tipos de datos de longitud variable.

Una alternativa a la asignación de espacio adicional o a la gestión del tipo de datos consiste en seleccionar los datos por fragmentos. Por ejemplo, para seleccionar el mismo VARCHAR(3000), que puede alcanzar 6000 bytes de longitud después de la conversión, puede realizar dos selecciones separadas, `SUBSTR(VC3000, 1, LENGTH(VC3000)/2)` y `SUBSTR(VC3000, (LENGTH(VC3000)/2)+1)`, en 2 áreas de aplicación VARCHAR(3000). Este método constituye la única solución posible cuando el tipo de datos ya no se puede gestionar. Por ejemplo, un CLOB codificado en la página de códigos DBCS del japonés con una longitud máxima de 2 gigabytes, posiblemente alcanzará dos veces su tamaño cuando se codifique en la página de códigos EUC del japonés. Esto significa que los datos se tendrán que dividir en fragmentos, puesto que no existe soporte para un tipo de datos cuya longitud supere 2 gigabytes.

Desbordamiento de longitud de serie de conversión de página de códigos en entornos de juegos de códigos mixtos

En entornos de páginas de códigos desiguales EUC y DBCS, después de que tenga lugar una conversión se pueden producir situaciones en que en una columna no haya suficiente espacio asignado para acomodar la serie entera. En este caso, la expansión máxima será dos veces la longitud de la serie en bytes. En los casos en que la expansión supera la capacidad de la columna, se devuelve SQLCODE -334 (SQLSTATE 22524).

Esto conduce a situaciones que pueden no resultar obvias de inmediato ni consideradas previamente:

- Una sentencia de SQL no puede tener una longitud superior a 32 765 bytes. Si la sentencia es suficientemente compleja o utiliza suficientes constantes o nombres de objetos que puedan estar sujetos a una expansión al convertirlos, se puede alcanzar este límite antes de lo esperado.
- Los identificadores de SQL tienen permitido expandirse al convertirlos hasta su longitud máxima, que es de ocho bytes para los identificadores cortos y de 128 bytes para los identificadores largos.
- Los identificadores de lenguaje principal tienen permitido expandirse al convertirlos hasta su longitud máxima, que es de 255 bytes.
- Cuando se convierten los campos de caracteres de la estructura SQLCA, sólo tienen permitido expandirse hasta su longitud máxima definida.

Cuando diseñe aplicaciones para entornos de juegos de códigos mixtos, debe consultar la documentación adecuada si se encuentra con una de las siguientes situaciones:

- Columnas de serie correspondientes en selecciones completas con operaciones de establecimiento (UNION, INTERSECT y EXCEPT)
- Operandos de concatenación
- Operandos de predicados (con la excepción de LIKE)
- Expresiones de resultados de una sentencia CASE
- Argumentos de la función escalar COALESCE (y VALUE)
- Valores de expresión de la lista IN de un predicado IN
- Expresiones correspondientes de una cláusula VALUES de varias filas

En estas situaciones, pueden tener lugar conversiones a la página de códigos de la aplicación en lugar de hacerlo a la página de códigos de la base de datos.

Otras situaciones que debe tener en cuenta son aquellas en las que la conversión de caracteres da como resultado una longitud de serie que sobrepasa el límite correspondiente al tipo de datos y conversiones de página de códigos en procedimientos almacenados:

- Conversión de caracteres sobrepasa un límite de tipo de datos

  En entornos de páginas de códigos desiguales EUC y DBCS, después de que tenga lugar una conversión se pueden producir situaciones en que la longitud de una serie de caracteres mixtos o gráfica supere la longitud máxima permitida para ese tipo de datos. Si la longitud de la serie, después de una expansión, supera el límite del tipo de datos, no se produce ninguna gestión del tipo. En cambio, se devuelve un mensaje de error que indica que se ha excedido la longitud máxima de expansión permitida. Es más probable que se produzca esta situación mientras se evalúan predicados que durante las inserciones. En las inserciones, la aplicación conoce más pronto la anchura de la columna y el factor de expansión máxima se puede tener en cuenta enseguida. En muchos casos, se puede evitar este efecto colateral de la conversión de caracteres cambiando el valor por un tipo de datos asociado cuya longitud máxima sea mayor. Por ejemplo, la longitud máxima de un valor CHAR es de 254 bytes, mientras que la longitud máxima de un VARCHAR es de 32 672 bytes. En los casos en que la expansión no supera la longitud máxima del tipo de datos, se devuelve SQLCODE -334 (SQLSTATE 22524).
- Conversión de página de códigos en un procedimiento almacenado

  Los datos de caracteres mixtos o gráficos especificados en variables de lenguaje principal y SQLDA en invocaciones a `sqleproc()` o a CALL de SQL se convierten en las situaciones en que las páginas de códigos de la aplicación y de

la base de datos son distintas. En los casos en que se produce una expansión de la longitud de la serie como consecuencia de la conversión, se recibe un SQLCODE -334 (SQLSTATE 22524) si no hay suficiente espacio asignado para manejar la expansión. Así, cuando cree procedimientos almacenados, se debe asegurar de proporcionar suficiente espacio para una expansión potencial de las series. Debe utilizar tipos de datos de longitud variable con un espacio asignado suficiente que permita la expansión.

Información relacionada:

- "Función escalar COALESCE" en la publicación *Consulta de SQL, Volumen 1*
- "Función escalar VALUE" en la publicación *Consulta de SQL, Volumen 1*
- "Selección completa" en la publicación *Consulta de SQL, Volumen 1*
- "Sentencia VALUES" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CASE" en la publicación *Consulta de SQL, Volumen 2*
- "Predicados" en la publicación *Consulta de SQL, Volumen 1*

Aplicaciones conectadas a bases de datos Unicode

Las aplicaciones de cualquier entorno de página de códigos se pueden conectar a una base de datos Unicode. Para las aplicaciones que se conectan a una base de datos Unicode, el gestor de bases de datos convierte los datos de serie de caracteres entre la página de códigos de la aplicación y la página de códigos de la base de datos (UTF-8). Cuando DB2 convierte caracteres de una página de códigos a UTF-8, el número total de bytes que representan los caracteres se puede expandir o reducir, según la página de códigos y los elementos de código de los caracteres. Los caracteres ASCII de 7 bits permanecen invariables en UTF-8, y cada carácter ASCII necesita un byte. Los caracteres que no son ASCII pasan a ocupar más de un byte cada uno. Para obtener más información sobre conversiones UTF-8, consulte los documentos estándar de Unicode.

Nota: La información que se aplica a las aplicaciones en código mixto también se aplica a las aplicaciones que se conectan a bases de datos Unicode.

Para una base de datos Unicode, los datos GRAPHIC están en orden UCS-2 big-endian. Si se utiliza el procesador de línea de mandatos para recuperar datos gráficos, los caracteres gráficos también se convierten a la página de códigos del cliente. Esta conversión permite que el procesador de línea de mandatos visualice caracteres gráficos en el font actual. Se puede producir una pérdida de datos siempre que el gestor de bases de datos convierta caracteres UCS-2 a la página de códigos de un cliente. Los caracteres que el gestor de bases de datos no puede convertir en caracteres válidos en la página de códigos del cliente se sustituyen por el carácter de sustitución por omisión de dicha página de códigos.

A partir de DB2® Versión 8, el gestor de bases de datos comprueba el valor de página de códigos del cliente y realiza todas las conversiones necesarias para los datos GRAPHIC UCS-2. Por ejemplo, si una aplicación que no sea Unicode envía datos GRAPHIC, DB2 convertirá los datos GRAPHIC a UCS-2 antes de que los datos se almacenen en una base de datos UCS-2. Y, a la inversa, si una aplicación que no sea Unicode solicita datos GRAPHIC de una base de datos UCS-2, DB2 convertirá los datos GRAPHIC a la página de códigos de la aplicación antes de que la aplicación acceda a los datos.

Nota: Se aplican las siguientes restricciones:

- Cuando los programas de utilidad de carga, importación o exportación de DB2 trabajan con un archivo DBCLOB, el gestor de bases de datos no comprueba la página de códigos del cliente.
- Cuando se recuperan datos GRAPHIC de una base de datos UCS-2 para una aplicación que no sea SBCS, EUC ni Unicode, DB2 sustituye por un carácter ASCII en blanco (U+0020) cada espacio en blanco que se rellena en la columna GRAPHIC UCS-2. La sustitución se lleva a cabo porque las páginas de códigos DBCS puras no tiene ningún equivalente al espacio en blanco UCS-2.
- Cuando se recuperan datos DATE, TIME y TIMESTAMP de una base de datos UCS-2 como datos del tipo GRAPHIC para una aplicación que no sea SBCS, EUC ni Unicode, DB2 convierte estos tipos de datos al carácter de sustitución. La sustitución se lleva a cabo porque los tipos de datos UCS-2 contienen caracteres SBCS que no tienen ningún equivalente en las páginas de códigos DBCS puras.

Antes de la Versión 8, DB2 no realizaba ninguna conversión automática de los datos GRAPHIC UCS-2. Eran las aplicaciones no-Unicode las que debían realizar las conversiones necesarias a Unicode y de Unicode o establecer la opción CONVERT de WCHARTYPE y utilizar `wchar_t`. Si un cliente de la Versión 7 se conecta a un servidor DB2 de la Versión 8, el gestor de bases de datos, por omisión, *no* lleva a cabo ninguna conversión de los datos para GRAPHIC UCS-2. Si desea alterar temporalmente este funcionamiento por omisión, puede establecer la variable del registro DB2GRAPHICUNICODESERVER en `OFF`.

Para las aplicaciones que se conectan a bases de datos DBCS, los datos GRAPHIC se convierten entre la página de códigos DBCS de la aplicación y la página de códigos DBCS de la base de datos.

Conceptos relacionados:

- "Unicode handling of data types" en la publicación *Administration Guide: Planning*
- "String comparisons in a Unicode database" en la publicación *Administration Guide: Planning*
- "Variables gráficas del lenguaje principal en C y C++" en la página 158
- "Consideraciones sobre nombres de paquetes en entornos de páginas de códigos combinadas" en la página 649
- "Consideraciones sobre las bases de datos y los clientes de doble byte con EUC mixtos" en la página 658
- "Validación de parámetros basada en cliente en un entorno de juegos de códigos mixtos" en la página 663
- "Sentencia DESCRIBE en entornos de juegos de códigos mixtos" en la página 664
- "Datos de longitud fija y de longitud variable en entornos de juegos de códigos mixtos" en la página 666
- "Desbordamiento de longitud de serie de conversión de página de códigos en entornos de juegos de códigos mixtos" en la página 666

Capítulo 30. Gestión de transacciones

Unidad de trabajo remota 671
Consideraciones sobre la actualización múltiple 671
Actualización múltiple 672
Cuándo utilizar la actualización múltiple . . . 672
Sentencias de SQL en aplicaciones de actualización múltiple 673
Precompilación de aplicaciones de actualización múltiple 674
Consideraciones sobre parámetros de configuración correspondientes a aplicaciones de actualización múltiple 676
Acceso a servidores de sistema principal, AS/400 o iSeries 677
Transacciones simultáneas 678
Transacciones simultáneas 678
Problemas potenciales con transacciones simultáneas 679
Cómo evitar puntos muertos para transacciones simultáneas 679
Puntos de rescate y transacciones. 680
Gestión de transacciones con puntos de rescate 680
Comparación entre puntos de rescate de la aplicación y bloques de SQL compuesto . . . 682
Sentencias de SQL para crear y controlar puntos de rescate 684
Restricciones sobre el uso de puntos de rescate 685
Puntos de rescate y Lenguaje de definición de datos (DDL) 685
Anidamiento de puntos de rescate 686
Puntos de rescate e inserciones en almacenamiento intermedio 687
Puntos de rescate y bloqueo de cursor 687
Puntos de rescate y gestores de transacciones que cumplen con XA 688
Consideraciones sobre la programación de interfaces XA de X/Open 688
Enlace de aplicaciones y la interfaz XA de X/Open 691
Gestión de transacciones MTS y COM+ 691
Microsoft Transaction Server (MTS) y Microsoft Component Services (COM+) como gestor de transacciones 691
Soporte ligeramente agrupado con Microsoft Component Services (COM+) 693
Tiempo de espera de transacciones de Microsoft Transaction Server (MTS) y Microsoft Component Services (COM+) 694
Agrupación de conexiones ODBC y ADO con Microsoft Transaction Server (MTS) y Microsoft Component Services (COM+) 695

Unidad de trabajo remota

Una unidad de trabajo es una sola transacción lógica. Consta de una secuencia de sentencias de SQL en que todas las operaciones se realizan satisfactoriamente o la secuencia global se considera no satisfactoria.

Una unidad de trabajo remota permite que un usuario o programa de aplicación lea o actualice datos que se encuentran en una ubicación por unidad de trabajo. Soporta el acceso a una base de datos dentro de una unidad de trabajo. Mientras que un programa de aplicación puede acceder a varias bases de datos remotas, sólo puede acceder a una base de datos de una unidad de trabajo.

Una unidad de trabajo remota tiene las características siguientes:
- Se soportan varias peticiones por unidad de trabajo.
- Se soportan varios cursores por unidad de trabajo.
- Cada unidad de trabajo sólo puede acceder a una base de datos.
- El programa de aplicación confirma o retrotrae la unidad de trabajo. En determinadas condiciones de error, el servidor puede retrotraer la unidad de trabajo.

Consideraciones sobre la actualización múltiple

Las secciones siguientes describen las actualizaciones múltiples y cómo desarrollar aplicaciones que realizan actualizaciones múltiples.

Actualización múltiple

Una actualización múltiple, también conocida como *unidad de trabajo distribuida* (DUOW) y como *confirmación en dos fases*, es una función que permite que las aplicaciones actualicen datos en varios servidores de bases de datos remotas con garantía de la integridad. Un buen ejemplo de actualización múltiple es una transacción bancaria que implica una transferencia monetaria de una cuenta a otra que se encuentra en otro servidor de bases de datos. En una transacción de este tipo, es vital que las actualizaciones que implementan la operación de cargo en una cuenta no se confirmen a no ser que también se confirmen las actualizaciones necesarias para procesar cargos en la otra cuenta. Las consideraciones sobre actualizaciones múltiples se aplican cuando los datos que representan estas cuentas son gestionados por dos servidores de bases de datos distintos.

Puede utilizar la actualización múltiple para leer y actualizar varias bases de datos DB2 Universal Database en una unidad de trabajo. Si ha instalado DB2® Connect o utiliza la funcionalidad de DB2 Connect™ que se suministra con DB2 Universal Database™ Enterprise Edition, también puede utilizar la actualización múltiple con servidores de bases de datos de sistema principal, de AS/400® o de iSeries, tales como DB2 Universal Database para z/OS y OS/390 y DB2 UDB para AS/400. Se aplican ciertas restricciones cuando se utiliza DB2 Connect en una actualización múltiple con otros servidores de bases de datos.

Un gestor de transacciones coordina la confirmación entre varias bases de datos. Si utiliza un entorno de supervisor de proceso de transacciones (TP) tal como TxSeries CICS®, el supervisor de TP utiliza su propio gestor de transacciones. De lo contrario, se utiliza el gestor de transacciones suministrado con DB2. Los sistemas operativos DB2 Universal Database para UNIX® y Windows® de 32 bits son gestores de recursos que cumplen la especificación XA (Extended Architecture). Los sistemas principales y los servidores de bases de datos iSeries a los que se accede con DB2 Connect son gestores de recursos que se ajustan a XA. Observe también que el gestor de transacciones de DB2 Universal Database *no* es un gestor de transacciones que se ajuste a XA, lo cual significa que el gestor de transacciones sólo puede coordinar bases de datos DB2.

Conceptos relacionados:

- "X/Open distributed transaction processing model" en la publicación *Administration Guide: Planning*
- "Multisite Updates" en la publicación *DB2 Connect User's Guide*

Cuándo utilizar la actualización múltiple

Una actualización múltiple es de la máxima utilidad cuando se desea trabajar con dos o más bases de datos y mantener la integridad de los datos. Por ejemplo, si cada sucursal de un banco tiene su propia base de datos, una aplicación de transferencia monetaria podría hacer lo siguiente:

1. Conectar con la base de datos del remitente.
2. Leer el saldo de la cuenta del remitente y verificar que tiene suficiente dinero.
3. Reducir el saldo de la cuenta del remitente en la cantidad transferida.
4. Conectar con la base de datos del receptor.
5. Aumentar el saldo de la cuenta del receptor con la cantidad transferida.
6. Confirmar las bases de datos.

Al realizar la transferencia de fondos dentro de una unidad de trabajo, se asegurará de que se actualicen ambas bases de datos o no se actualice ninguna de ellas.

Sentencias de SQL en aplicaciones de actualización múltiple

La tabla siguiente le muestra cómo codificar sentencias de SQL para una actualización múltiple. La columna de la izquierda muestra sentencias de SQL que no utilizan la actualización múltiple; la de la derecha, muestra sentencias parecidas que contienen una actualización múltiple.

Tabla 92. Sentencias de RUOW y de SQL con una actualización múltiple

Sentencias de RUOW	Sentencias de actualización múltiple

CONNECT TO D1
  SELECT
  UPDATE
  COMMIT
 	
CONNECT TO D2
  INSERT
  COMMIT
 	
CONNECT TO D1
  SELECT
  COMMIT
CONNECT RESET
	CONNECT TO D1
	  SELECT
	  UPDATE
	
	CONNECT TO D2
	  INSERT
	  RELEASE CURRENT
	
	SET CONNECTION D1
	  SELECT
	  RELEASE D1
	COMMIT

Las sentencias de SQL de la columna de la izquierda sólo acceden a una base de datos por cada unidad de trabajo. Ésta es una aplicación de unidad de trabajo remota (RUOW).

Las sentencias de SQL de la columna de la derecha acceden a más de una base de datos en una unidad de trabajo. Ésta es una aplicación de actualización múltiple.

Algunas sentencias de SQL se codifican e interpretan de forma distinta en una aplicación de actualización múltiple:

- No es necesario confirmar ni retrotraer la unidad de trabajo actual antes de conectar con otra base de datos.
- Cuando se conecta con otra base de datos, no se desconecta la conexión actual. En cambio, se pone en estado *latente*. Si la sentencia CONNECT falla, la conexión actual no se ve afectada.
- No se puede conectar con la cláusula USER/USING si ya existe una conexión actual o latente con la base de datos.
- Puede utilizar la sentencia SET CONNECTION para hacer que una conexión latente pase a ser la conexión actual.

  Se puede lograr el mismo efecto emitiendo una sentencia CONNECT para la base de datos latente. Este método no está permitido si SQLRULES se establece en STD. Puede establecer el valor de SQLRULES utilizando una opción de precompilador, el mandato SET CLIENT o la API. El valor por omisión de SQLRULES (DB2) permite conmutar conexiones mediante la sentencia CONNECT.
- En una selección, la posición del cursor no se ve afectada si se conmuta a otra base de datos y luego se vuelve a la base de datos original.

- La sentencia CONNECT RESET no desconecta la conexión actual y no confirma implícitamente la unidad de trabajo actual. En cambio, esta sentencia es equivalente a una conexión explícita con la base de datos por omisión (en caso de que se haya definido). Si no se define una conexión implícita, se devuelve SQLCODE -1024 (SQLSTATE 08003).
- Puede utilizar la sentencia RELEASE para marcar una conexión de forma que se desconecte en la próxima COMMIT. La sentencia RELEASE CURRENT se aplica a la conexión actual, la sentencia RELEASE *conexión* se aplica a la conexión mencionada y la sentencia RELEASE ALL se aplica a todas las conexiones.

  Una conexión que está marcada para ser liberada se puede seguir utilizando hasta que se desactive en la próxima sentencia COMMIT. Una retrotracción no desactiva la conexión; este comportamiento permite efectuar un reintento con las conexiones establecidas. Utilice la sentencia DISCONNECT (o una opción de precompilador) para desactivar las conexiones después de una confirmación o retrotracción.
- La sentencia COMMIT confirma todas las bases de datos de la unidad de trabajo (actual o latente).
- La sentencia ROLLBACK retrotrae todas las bases de datos de la unidad de trabajo y cierra los cursores retenidos para todas las bases de datos, tanto si se accede a ellos en la unidad de trabajo como si no es así.
- Todas las conexiones (incluidas las latentes y las marcadas para su liberación) se desconectan cuando termina el proceso de la aplicación.
- Después de cualquier conexión satisfactoria (incluyendo una sentencia CONNECT sin opciones, que sólo consulta la conexión actual) se devolverá un número en los campos SQLERRD(3) y SQLERRD(4) de la SQLCA.

  El campo SQLERRD(3) devuelve información sobre si la base de datos con la que se ha conectado se puede actualizar actualmente en una unidad de trabajo. Los valores posibles de este campo son:

  1 Se puede actualizar.
  2 Sólo es de lectura.

  El campo SQLERRD(4) devuelve la información siguiente sobre las características actuales de la conexión:

  0 No es aplicable. Este estado sólo es posible si se ejecuta desde un cliente de nivel inferior que utiliza la confirmación en una fase y es un actualizador.

  1 Confirmación en una fase.

  2 Confirmación en una fase (sólo lectura). Este estado sólo se aplica a servidores de bases de datos de sistema principal, AS/400® o iSeries a los que accede con DB2® Connect *sin* iniciar el gestor de puntos de sincronismo de DB2 Connect™.

  3 Confirmación de dos fases.

  Si está escribiendo herramientas o programas de utilidad, es posible que desee emitir un mensaje a los usuarios en caso de que la conexión sólo sea de lectura.

Precompilación de aplicaciones de actualización múltiple

Cuando precompile una aplicación de actualización múltiple, debe establecer la conexión del CLP en una conexión de tipo 1; de lo contrario, recibirá un SQLCODE 30090 (SQLSTATE 25000) cuando intente precompilar la aplicación. Cuando se precompila una aplicación que utiliza actualizaciones múltiples, se utilizan las opciones de precompilador siguientes:

CONNECT (1 | 2)
Especifique CONNECT 2 para indicar que esta aplicación utiliza la sintaxis de SQL para las aplicaciones de actualización múltiple. El valor por omisión, CONNECT 1, significa que se aplican a la aplicación las reglas normales (RUOW) para la sintaxis de SQL.

SYNCPOINT (ONEPHASE | TWOPHASE | NONE)
Si especifica SYNCPOINT TWOPHASE y DB2® coordina la transacción, DB2 necesita una base de datos para mantener la información de estado de la transacción. Cuando despliegue la aplicación, debe definir esta base de datos configurando el parámetro de configuración del gestor de bases de datos *tm_database*.

SQLRULES (DB2 | STD)
Especifica si en las aplicaciones de actualización múltiple se deben utilizar las reglas de DB2 o las reglas estándar (STD) basadas en ISO/ANSI SQL92. Las reglas de DB2 permiten emitir una sentencia CONNECT para una base de datos latente; las reglas STD no lo permiten.

DISCONNECT (EXPLICIT | CONDITIONAL | AUTOMATIC)
Especifica qué conexiones de base de datos se deben desconectar cuando se ejecute COMMIT: sólo las bases de datos que están marcadas para su liberación con una sentencia RELEASE (EXPLICIT), todas las bases de datos que no tienen cursores WITH HOLD abiertos (CONDITIONAL) o todas las conexiones (AUTOMATIC).

Las opciones de precompilador de la actualización múltiple entran en vigor cuando se realiza la primera conexión con una base de datos. Puede utilizar la API SET CLIENT para sustituir los valores de conexión cuando no exista ninguna conexión (antes de que se establezca alguna conexión o después de que se desconecten todas ellas). Puede utilizar la API QUERY CLIENT para consular los valores de conexión actuales del proceso de la aplicación.

Si un objeto al que se hace referencia en el programa de aplicación no existe, el enlazador falla. Existen tres maneras posibles de tratar las aplicaciones de actualización múltiple:

- Puede dividir la aplicación en varios archivos, cada uno de los cuales accederá a una sola base de datos. Luego prepare y vincule cada archivo a la única base de datos a la que acceda.
- Puede asegurarse de que exista cada tabla en cada base de datos. Por ejemplo, en las sucursales de un banco pueden haber bases de datos cuyas tablas sean idénticas (a excepción de los datos).
- Puede utilizar únicamente SQL dinámico.

Conceptos relacionados:

- "Sentencias de SQL en aplicaciones de actualización múltiple" en la página 673

Información relacionada:

- "Sentencia CONNECT (Tipo 1)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CONNECT (Tipo 2)" en la publicación *Consulta de SQL, Volumen 2*
- "sqlesetc - Set Client" en la publicación *Administrative API Reference*
- "sqleqryi - Query Client Information" en la publicación *Administrative API Reference*
- "Mandato PRECOMPILE" en la publicación *Consulta de mandatos*

Consideraciones sobre parámetros de configuración correspondientes a aplicaciones de actualización múltiple

Los parámetros de configuración siguientes afectan a las aplicaciones que realizan actualizaciones múltiples. A excepción de *locktimeout*, los parámetros de configuración son parámetros de configuración del gestor de bases de datos. *locktimeout* es un parámetro de configuración de la base de datos.

tm_database
: Especifica qué base de datos va a actuar como gestor de transacciones para las confirmaciones de dos fases.

resync_interval
: Especifica el número de segundos que el sistema esperará entre intentos para tratar de resincronizar una transacción dudosa. (Una transacción dudosa es una transacción que completa satisfactoriamente la primera fase de una confirmación en dos fases pero falla durante la segunda fase.)

locktimeout
: Especifica el número de segundos antes de que se exceda el tiempo de espera de un bloqueo y se retrotraiga la transacción actual para una base de datos determinada. La aplicación debe emitir una sentencia ROLLBACK explícita para retrotraer todas las bases de datos que participan en la actualización múltiple. *locktimeout* es un parámetro de configuración de la base de datos.

tp_mon_name
: Especifica el nombre del supervisor de TP, si lo hay.

spm_resync_agent_limit
: Especifica el número de agentes simultáneos que pueden realizar operaciones de resincronización con el servidor de sistema principal, AS/400® o iSeries utilizando SNA.

spm_name

- Si se utiliza el gestor de puntos de sincronismo con una conexión TCP/IP de confirmación en dos fases, *spm_name* debe ser un identificador exclusivo dentro de la red. Cuando se crea una instancia de DB2®, DB2 deduce el valor por omisión de *spm_name* a partir del nombre de sistema principal TCP/IP. El usuario puede modificar este valor si no resulta aceptable en su entorno. Para la conectividad de TCP/IP con los servidores de bases de datos de sistema principal, el valor por omisión debe resultar aceptable. Para las conexiones SNA con servidores de bases de datos de sistema principal, AS/400 o iSeries, este valor debe coincidir con un perfil de LU SNA definido en el producto SNA.
- Si se utiliza el gestor de puntos de sincronismo con una conexión SNA de confirmación en dos fases, el nombre del gestor de puntos de sincronismo debe tener el valor LU_NAME que se utiliza para la confirmación en dos fases.
- Si se utiliza el gestor de puntos de sincronismo tanto para TCP/IP como para SNA, se debe utilizar el valor de LU_NAME que se utiliza para la confirmación en dos fases.

Nota: Las actualizaciones múltiples en un entorno con servidores de bases de datos de sistema principal, AS/400 o iSeries pueden necesitar el gestor de puntos de sincronismo.

spm_log_size
El número de páginas de 4 KB de cada archivo de registro primario y secundario utilizado por el gestor de puntos de sincronismo para registrar información sobre las conexiones, el estado de las conexiones actuales, etc.

Hay otras consideraciones que deben tenerse en cuenta si la aplicación realiza actualizaciones múltiples que coordina un gestor de transacciones XA con conexiones a una base de datos de sistema principal, AS/400 o iSeries.

Conceptos relacionados:

- "Multisite Updates" en la publicación *DB2 Connect User's Guide*
- "Multisite update and sync point manager" en la publicación *DB2 Connect User's Guide*

Tareas relacionadas:

- "Enabling Multisite Updates using the Control Center" en la publicación *DB2 Connect User's Guide*

Información relacionada:

- "spm_log_path - Sync point manager log file path configuration parameter" en la publicación *Administration Guide: Performance*
- "resync_interval - Transaction resync interval configuration parameter" en la publicación *Administration Guide: Performance*
- "tm_database - Transaction manager database name configuration parameter" en la publicación *Administration Guide: Performance*
- "tp_mon_name - Transaction processor monitor name configuration parameter" en la publicación *Administration Guide: Performance*
- "locktimeout - Lock timeout configuration parameter" en la publicación *Administration Guide: Performance*
- "spm_name - Sync point manager name configuration parameter" en la publicación *Administration Guide: Performance*
- "spm_log_file_sz - Sync point manager log file size configuration parameter" en la publicación *Administration Guide: Performance*
- "spm_max_resync - Sync point manager resync agent limit configuration parameter" en la publicación *Administration Guide: Performance*

Acceso a servidores de sistema principal, AS/400 o iSeries

Procedimiento:

Si desea crear aplicaciones que accedan a sistemas de bases de datos diferentes (o los actualicen), debe:

1. Utilizar sentencias de SQL y opciones de precompilación/vinculación que se soporten en todos los sistemas de bases de datos a los que vayan a acceder las aplicaciones. Por ejemplo, los procedimientos almacenados no se soportan en todas las plataformas.

   Para productos IBM, consulte la documentación de SQL *antes* de empezar a codificar.
2. Si es posible, hacer que las aplicaciones comprueben SQLSTATE en lugar de SQLCODE.

Si las aplicaciones utilizarán DB2 Connect y desea utilizar valores SQLCODE, considere la posibilidad de utilizar el recurso de correlación que brinda DB2 Connect para correlacionar las conversiones de SQLCODE entre bases de datos distintas.

3. Pruebe la aplicación con las bases de datos de sistema principal, AS/400 o iSeries (tales como DB2 Universal Database para z/OS y OS/390, OS/400 o DB2 Server para VSE y VM) que pretenda soportar.

Conceptos relacionados:

- "Aplicaciones en entornos de sistema principal o iSeries" en la página 739

Transacciones simultáneas

Las secciones siguientes describen las transacciones simultáneas y cómo evitar problemas con las mismas.

Transacciones simultáneas

Algunas veces, resulta de utilidad para una aplicación tener varias conexiones independientes denominadas *transacciones simultáneas*. Mediante la utilización de transacciones simultáneas, una aplicación puede conectar con varias bases de datos al mismo tiempo, y puede establecer varias conexiones distintas con la misma base de datos.

Las API de contexto que se utilizan para el acceso a bases de datos de varias hebras permiten que una aplicación utilice transacciones simultáneas. Cada contexto creado en una aplicación es independiente de los otros contextos. Esto significa que se crea un contexto, se conecta con una base de datos utilizando el contexto y se ejecutan sentencias de SQL para la base de datos sin que se vean afectadas por actividades de otros contextos, tales como la ejecución de sentencias COMMIT o ROLLBACK.

Por ejemplo, suponga que está creando una aplicación que permite que un usuario ejecute sentencias de SQL para una base de datos, y que mantiene un registro de las actividades realizadas en una segunda base de datos. Puesto que el registro se debe mantener actualizado, es necesario emitir una sentencia COMMIT después de cada actualización del mismo, pero no se desea que las sentencias de SQL del usuario se vean afectadas por las confirmaciones del registro. Ésta es una situación perfecta para las transacciones simultáneas. En la aplicación, cree dos contextos: uno conectará con la base de datos del usuario y se utilizará para todo el SQL del usuario; el otro conectará con la base de datos del registro y se utilizará para actualizar el registro. Mediante este diseño, cuando confirme un cambio efectuado en la base de datos del registro, no afectará a la unidad de trabajo actual del usuario.

Otro beneficio de las transacciones simultáneas es que, si se retrotrae el trabajo sobre los cursores de una conexión, esto no tiene ningún efecto sobre los cursores de otras conexiones. Después de la retrotracción de la primera conexión, en las otras conexiones se mantienen tanto el trabajo realizado como las posiciones de los cursores.

Conceptos relacionados:

- "Objetivo del acceso a base de datos de varias hebras" en la página 187

Problemas potenciales con transacciones simultáneas

Una aplicación que utiliza transacciones simultáneas se puede encontrar con algunos problemas que no pueden surgir cuando se escribe una aplicación que utiliza una sola conexión. Cuando escriba una aplicación con transacciones simultáneas, tenga cuidado con los aspectos siguientes:

- Dependencias de base de datos entre dos o más contextos.

  Cada contexto de una aplicación tiene sus propios recursos de base de datos, incluidos los bloqueos sobre objetos de base de datos. Estos conjuntos distintos de recursos posibilitan que dos contextos, si están accediendo al mismo objeto de base de datos, lleguen a un punto muerto. El gestor de bases de datos detectará el punto muerto, uno de los contextos recibirá SQLCODE -911 y se retrotraerá la unidad de trabajo del mismo.

- Dependencias de aplicaciones entre dos o más contextos.

  La conmutación de contextos dentro de una sola hebra crea dependencias entre los contextos. Si los contextos también tienen dependencias de base de datos, es posible que se produzca un punto muerto. Puesto que algunas de las dependencias se producen fuera del gestor de bases de datos, no se detectará el punto muerto y se suspenderá la aplicación.

  Como ejemplo de este tipo de problema, piense en la aplicación siguiente:

```
   contexto 1
   UPDATE TAB1 SET COL = :new_val

   contexto 2
   SELECT * FROM TAB1
   COMMIT

   contexto 1
COMMIT
```

  Suponga que el primer contexto ejecuta satisfactoriamente la sentencia UPDATE. La actualización establece bloqueos sobre todas las filas de TAB1. Ahora, el contexto 2 intenta seleccionar todas las filas de TAB1. Puesto que los dos contextos son independientes, el contexto 2 espera a causa de los bloqueos mantenidos por el contexto 1. Sin embargo, el contexto 1 no puede liberar sus bloqueos hasta que termina la ejecución del contexto 2. La aplicación está ahora en un punto muerto, pero el gestor de bases de datos no sabe que el contexto está esperando al contexto 2, por lo que no impondrá que se retrotraiga uno de los contextos. La dependencia no resuelta deja la aplicación suspendida.

Conceptos relacionados:

- "Cómo evitar puntos muertos para transacciones simultáneas" en la página 679

Cómo evitar puntos muertos para transacciones simultáneas

Puesto que el gestor de bases de datos no puede detectar puntos muertos entre contextos, debe diseñar y codificar la aplicación de forma que impida (o, como mínimo, permita que se eviten) los puntos muertos. Supongamos que tenemos el siguiente ejemplo, que puede dar lugar a una situación de punto muerto:

```
contexto 1
UPDATE TAB1 SET COL = :new_val

contexto 2
SELECT * FROM TAB1
```

```
COMMIT

contexto 1
COMMIT
```

Suponga que el primer contexto ejecuta satisfactoriamente la sentencia UPDATE. La actualización establece bloqueos sobre todas las filas de TAB1. Ahora, el contexto 2 intenta seleccionar todas las filas de TAB1. Puesto que los dos contextos son independientes, el contexto 2 espera a causa de los bloqueos mantenidos por el contexto 1. Sin embargo, el contexto 1 no puede liberar sus bloqueos hasta que termina la ejecución del contexto 2. La aplicación está ahora en un punto muerto, pero el gestor de bases de datos no sabe que el contexto está esperando al contexto 2, por lo que no impondrá que se retrotraiga uno de los contextos. La dependencia no resuelta deja la aplicación suspendida.

Puede evitar el punto muerto del ejemplo de varias formas:

- Libere todos los bloqueos mantenidos antes de conmutar contextos.

  Cambie el código de forma que el contexto 1 realice su confirmación antes de conmutar al contexto 2.
- No acceda a un objeto determinado desde más de un contexto a la vez.

  Cambie el código de forma que tanto la actualización como la selección se realicen desde el mismo contexto.
- Establezca el parámetro de configuración de la base de datos *locktimeout* en un valor distinto de -1.

  Aunque un valor distinto de -1 no impedirá el punto muerto, permitirá que se reanude la ejecución. Eventualmente, se retrotraerá el contexto 2, puesto que no puede obtener el bloqueo solicitado. Cuando se retrotraiga el contexto 2, el contexto 1 podrá continuar la ejecución (con lo cual liberará los bloqueos) y el contexto 2 podrá reintentar su trabajo.

Aunque las técnicas para evitar puntos muertos se describen en términos del ejemplo, las puede aplicar a todas las aplicaciones que utilizan transacciones simultáneas.

Conceptos relacionados:

- "Problemas potenciales con transacciones simultáneas" en la página 679

Información relacionada:

- "locktimeout - Lock timeout configuration parameter" en la publicación *Administration Guide: Performance*

Puntos de rescate y transacciones

Las secciones siguientes describen los puntos de rescate y cómo utilizarlos para gestionar transacciones.

Gestión de transacciones con puntos de rescate

Los puntos de rescate de aplicaciones proporcionan control sobre el trabajo realizado por un subconjunto de sentencias de SQL en una transacción o unidad de trabajo. Dentro de la aplicación puede definir un punto de rescate y luego liberarlo o retrotraer el trabajo realizado desde que definió el punto de rescate. Puede utilizar tantos puntos de rescate como sea necesario dentro de una transacción

individual. El siguiente ejemplo muestra el uso de dos puntos de rescate dentro de una sola transacción para controlar el comportamiento de una aplicación:

Ejemplo de un pedido que utiliza puntos de rescate de la aplicación:

```
INSERT INTO order ...
INSERT INTO order_item ... lamp

-- definir el primer punto de rescate de la transacción
SAVEPOINT before_radio ON ROLLBACK RETAIN® CURSORS
INSERT INTO order_item ... Radio
INSERT INTO order_item ... Power Cord
-- Pseudo-SQL:
IF SQLSTATE = "No Power Cord"
    ROLLBACK TO SAVEPOINT before_radio
RELEASE SAVEPOINT before_radio

-- definir el segundo punto de rescate de la transacción
SAVEPOINT before_checkout ON ROLLBACK RETAIN CURSORS
INSERT INTO order ... Approval
-- Pseudo-SQL:
IF SQLSTATE = "No approval"
    ROLLBACK TO SAVEPOINT before_checkout

-- confirmar la transacción, lo cual libera el punto de rescate
COMMIT
```

En el ejemplo anterior, el primer punto de rescate hace cumplir una dependencia entre dos objetos de datos en los que la dependencia no es intrínseca de los propios objetos. No utilizaría integridad referencial para describir la relación anterior entre radios y cables de alimentación (power cords) puesto que un objeto puede existir sin el otro. Sin embargo, no desea distribuir al cliente la radio sin un cable de alimentación. Tampoco desea cancelar el pedido de la lámpara (lamp) retrotrayendo la transacción entera porque no hay cables de alimentación para la radio. Los puntos de rescate de la aplicación proporcionan el control granular que necesita para completar este pedido.

Cuando emite una sentencia ROLLBACK TO SAVEPOINT, el punto de rescate correspondiente no se libera automáticamente. Las siguientes sentencias de SQL se asocian con este punto de rescate hasta que este se libera de forma explícita con una sentencia RELEASE SAVEPOINT o de forma implícita finalizando la transacción o unidad de trabajo. El punto de rescate también se puede liberar implícitamente al final del nivel del punto de rescate actual o hasta que se libere o retrotraiga un punto de rescate activo anterior, en cuyo momento el punto de rescate actual pasará a ser obsoleto. Esto significa que puede emitir varias sentencias ROLLBACK TO SAVEPOINT para un solo punto de rescate.

Los puntos de rescate le ofrecen un mejor rendimiento y un diseño de aplicación más limpio que la utilización de varias sentencias COMMIT y ROLLBACK. Cuando emite una sentencia COMMIT, DB2® debe realizar un trabajo adicional para confirmar la transacción actual e iniciar una nueva transacción. Los puntos de rescate le permiten dividir una transacción en unidades de menor tamaño o pasos sin el proceso general añadido de varias sentencias COMMIT. El siguiente ejemplo muestra el deterioro del rendimiento que se produce si se utilizan varias transacciones en lugar de puntos de rescate:

Ejemplo de un pedido que utiliza varias transacciones::

```
INSERT INTO order ...
INSERT INTO order_item ... lamp
  -- confirmar transacción actual, iniciar transacción nueva
```

```
COMMIT
INSERT INTO order_item ... Radio
INSERT INTO order_item ... Power Cord
-- Pseudo-SQL:
IF SQLSTATE = "No Power Cord"
  -- retrotraer transacción actual, iniciar transacción nueva
  ROLLBACK
ELSE
  -- confirmar transacción actual, iniciar transacción nueva
  COMMIT

INSERT INTO order ... Approval
-- Pseudo-SQL:
IF SQLSTATE = "No approval"
  -- retrotraer transacción actual, iniciar transacción nueva
  ROLLBACK
ELSE
  -- confirmar transacción actual, iniciar transacción nueva
COMMIT
```

Otro inconveniente de utilizar varios puntos de confirmación es que se puede confirmar un objeto, con lo cual estará visible para otras aplicaciones antes de que haya finalizado por completo. En el segundo ejemplo, el pedido está disponible para otro usuario antes de que se hayan añadido todos los elementos y, lo que es peor, antes de que se haya aprobado. El uso de puntos de rescate de la aplicación evita esta exposición a 'datos sucios' mientras se ofrece un control granular sobre una operación.

Ejemplos relacionados:

- "tbsavept.sqc -- How to use external savepoints (C)"

Comparación entre puntos de rescate de la aplicación y bloques de SQL compuesto

Los puntos de rescate ofrecen las siguientes ventajas sobre los bloques de SQL compuesto:

- Control mejorado de transacciones
- Menor contención de bloqueo
- Integración mejorada con la lógica de la aplicación

Los bloques de SQL compuesto pueden ser de tipo ATOMIC o NOT ATOMIC. Si una sentencia dentro de un bloque de SQL compuesto de tipo ATOMIC falla, el bloque de SQL compuesto entero se retrotrae. Si una sentencia dentro de un bloque de SQL compuesto de tipo NOT ATOMIC falla, la aplicación controla la confirmación o retrotracción de la aplicación, incluido el bloque de SQL compuesto entero. En comparación, si una sentencia dentro del ámbito de un punto de rescate falla, la aplicación puede retrotraer todas las sentencias del ámbito del punto de rescate pero confirmar el trabajo realizado por sentencias externas al ámbito del punto de rescate. Esta opción se ilustra en el siguiente ejemplo. Si el trabajo del punto de rescate se retrotrae, el trabajo de las dos sentencias INSERT anteriores al punto de rescate se confirma. Como alternativa, la aplicación puede confirmar el trabajo realizado por todas las sentencias de la transacción, incluidas las sentencias dentro del ámbito del punto de rescate.

Ejemplo de un pedido que utiliza puntos de rescate de la aplicación:

```
INSERT INTO order ...
INSERT INTO order_item ... lamp
```

```
-- definir el primer punto de rescate de la transacción
SAVEPOINT before_radio ON ROLLBACK RETAIN® CURSORS
    INSERT INTO order_item ... Radio
    INSERT INTO order_item ... Power Cord
    -- Pseudo-SQL:
    IF SQLSTATE = "No Power Cord"
      ROLLBACK TO SAVEPOINT before_radio
RELEASE SAVEPOINT before_radio

-- definir el segundo punto de rescate de la transacción
SAVEPOINT before_checkout ON ROLLBACK RETAIN CURSORS
  INSERT INTO order ... Approval
    -- Pseudo-SQL:
  IF SQLSTATE = "No approval"
    ROLLBACK TO SAVEPOINT before_checkout

-- confirmar la transacción, lo cual libera el punto de rescate
COMMIT
```

Cuando emite un bloque de SQL compuesto, DB2® adquiere simultáneamente los bloqueos necesarios para el bloque completo de sentencias de SQL compuesto. Cuando define un punto de rescate de la aplicación, DB2 adquiere bloqueos a medida que se emite cada sentencia en el ámbito del punto de rescate. Este comportamiento de bloqueo de puntos de rescate puede llevar a una contención de bloqueo significativamente menor que la que ocasionan los bloques de SQL compuesto, de modo que, a menos que la aplicación necesite que el bloqueo lo realicen las sentencias de SQL compuesto, puede ser aconsejable utilizar puntos de rescate.

Los bloques de SQL compuesto ejecutan un conjunto completo de sentencias como una sola sentencia. Una aplicación no puede utilizar funciones o estructuras de control para añadir sentencias a un bloque de SQL compuesto. En comparación, cuando define un punto de rescate de la aplicación, esta puede emitir sentencias de SQL dentro del ámbito del punto de rescate realizando llamadas a otros métodos o funciones de aplicación, a través de estructuras de control como bucles while o con sentencias de SQL dinámico. Los puntos de rescate de la aplicación le ofrecen libertad para integrar las sentencias de SQL con la lógica de la aplicación de forma intuitiva.

Por ejemplo, en el ejemplo siguiente la aplicación define un punto de rescate y emite dos sentencias INSERT dentro del ámbito del punto de rescate. La aplicación utiliza una sentencia IF que, cuando es cierta, llama a la función add_batteries(). La función add_batteries() emite un sentencia de SQL que en este contexto se incluye dentro del ámbito del punto de rescate. Finalmente, la aplicación retrotrae el trabajo realizado dentro del punto de rescate (incluida la sentencia de SQL que ha emitido la función add_batteries()) o confirma el trabajo realizado en la transacción entera:

Ejemplo de integración de puntos de rescate y sentencias de SQL dentro de la lógica de la aplicación:

```
void add_batteries()
{
  -- el punto de rescate definido en main() controla
  -- el trabajo realizado por la siguiente sentencia
  INSERT INTO order_item ... Batteries
}

void main(int argc, char[] *argv)
{
  INSERT INTO order ...
```

```
  INSERT INTO order_item ... lamp

  -- definir el primer punto de rescate de la transacción
  SAVEPOINT before_radio ON ROLLBACK RETAIN CURSORS
    INSERT INTO order_item ... Radio
    INSERT INTO order_item ... Power Cord

    if (strcmp(Radio..power_source(), "AC/DC"))
    {
      add_batteries();
    }

    -- Pseudo-SQL:
    IF SQLSTATE = "No Power Cord"
      ROLLBACK TO SAVEPOINT before_radio
  COMMIT
}
```

Sentencias de SQL para crear y controlar puntos de rescate

Las siguientes sentencias de SQL le permiten crear y controlar puntos de rescate:

SAVEPOINT
Para definir un punto de rescate, emita una sentencia de SQL SAVEPOINT. Para identificar un punto de rescate determinado de entre varios puntos de rescate anidados y mejorar la legibilidad del código de programación, puede asignar un nombre descriptivo al punto de rescate. Estos nombres se denominan referencias de punto de rescate. Por ejemplo:

```
SAVEPOINT before_sales ON ROLLBACK RETAIN CURSORS
```

RELEASE SAVEPOINT
Para liberar un punto de rescate, emita una sentencia de SQL RELEASE SAVEPOINT. Por ejemplo:

```
RELEASE SAVEPOINT before_sales
```

Si no libera explícitamente un punto de rescate con una sentencia RELEASE SAVEPOINT de SQL, se libera al final del nivel actual de punto de rescate.

ROLLBACK TO SAVEPOINT
Para retrotraer un punto de rescate, emita una sentencia de SQL ROLLBACK TO SAVEPOINT. Por ejemplo:

```
ROLLBACK TO SAVEPOINT before_sales
```

Un nivel de punto de rescate es el ámbito de referencia de una sentencia que hace uso de puntos de rescate. Cuando se inicia un nivel de punto de rescate, ninguna sentencia que haga uso de puntos de rescate puede hacer referencia a un punto de rescate creado fuera del nuevo nivel de punto de rescate. Similarmente, las referencias a puntos de rescate se resuelven dentro del nivel actual de punto de rescate y no tienen en cuenta las referencias a puntos de rescate situadas fuera del nivel actual de punto de rescate.

Se inicia un nuevo nivel de punto de rescate o se entra en él sólo cuando se produce una de estas condiciones:

- Se inicia una nueva unidad de trabajo
- Se ha definido un procedimiento almacenado con la cláusula NEW SAVEPOINT LEVEL
- Se inicia un sentencia atómica compuesta de SQL

Un nivel de punto de rescate finaliza cuando finaliza o se elimina el suceso que causó su creación.

Son aplicables las reglas siguientes para las acciones realizadas dentro del ámbito de un nivel de punto de rescate:

- Solo se pueden referenciar puntos de rescate dentro del nivel de punto de rescate en el que están definidos. No puede liberar ni retrotraer hacia un punto de rescate establecido fuera del nivel actual de punto de rescate.
- Todos los puntos de rescate activos establecidos dentro del nivel actual de punto de rescate se liberan automáticamente cuando finaliza el nivel de punto de rescate.
- Los nombres de punto de rescate exclusivos solo se aplican dentro del nivel actual de punto de rescate. Los nombres de puntos de rescate que están activos en los niveles circundantes se pueden reutilizar en el nivel actual sin afectar a esos otros puntos de rescate.

Información relacionada:

- "Sentencia ROLLBACK" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia RELEASE SAVEPOINT" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia SAVEPOINT" en la publicación *Consulta de SQL, Volumen 2*

Ejemplos relacionados:

- "tbsavept.sqc -- How to use external savepoints (C)"

Restricciones sobre el uso de puntos de rescate

DB2® Universal Database aplica las restricciones siguientes respecto a la utilización de puntos de rescate en las aplicaciones:

Activadores
DB2 no permite utilizar sentencias con puntos de rescate dentro de la definición de un activador. Pero puede utilizar un activador para invocar procedimientos almacenados que contengan puntos de rescate. Cuando se invocan, estos procedimientos almacenados deben iniciar un nuevo nivel de punto de rescate (esto se realiza con la cláusula NEW SAVEPOINT LEVEL de la sentencia CREATE PROCEDURE).

sentencia SET INTEGRITY
Dentro de un punto de rescate, DB2 trata las sentencias SET INTEGRITY como sentencias de DDL.

Conceptos relacionados:

- "Puntos de rescate y Lenguaje de definición de datos (DDL)" en la página 685

Puntos de rescate y Lenguaje de definición de datos (DDL)

DB2® le permite incluir sentencias de DDL dentro de un punto de rescate. Si la aplicación libera satisfactoriamente un punto de rescate que ejecuta sentencias de DDL, la aplicación puede continuar utilizando los objetos SQL creados por DDL. Sin embargo, si la aplicación emite una sentencia ROLLBACK TO SAVEPOINT para un punto de rescate que ejecuta sentencias de DDL, DB2 marca los cursores que dependen de los efectos de dichas sentencias de DDL como no válidos.

En el siguiente ejemplo, la aplicación intenta captar desde de los tres cursores abiertos anteriormente después de emitir una sentencia ROLLBACK TO SAVEPOINT:

```
SAVEPOINT savepoint_name;
   PREPARE s1 FROM 'SELECT FROM t1';
   --emitir sentencia de DDL para t1
      ALTER TABLE t1 ADD COLUMN...
   PREPARE s2 FROM 'SELECT FROM t2';
   --emitir sentencia de DDL para t3
      ALTER TABLE t3 ADD COLUMN...
   PREPARE s3 FROM 'SELECT FROM t3';
   OPEN c1 USING s1;
   OPEN c2 USING s2;
   OPEN c3 USING s3;
ROLLBACK TO SAVEPOINT
FETCH c1; --no válido (SQLCODE -910)
FETCH c2; --satisfactorio
FETCH c3; --no válido (SQLCODE -910)
```

En la sentencia ROLLBACK TO SAVEPOINT, DB2 marca los cursores “c1” y “c3” como no válidos porque las sentencias de DDL dentro del punto de rescate han manipulado los objetos SQL de los que dependen. Sin embargo, una sentencia FETCH que utiliza el cursor “c2” en el ejemplo es satisfactoria después de la sentencia ROLLBACK TO SAVEPOINT.

Puede emitir una sentencia CLOSE para cerrar los cursores no válidos. Si emite una sentencia FETCH contra un cursor no válido, DB2 devuelve SQLCODE –910. Si emite una sentencia OPEN contra un cursor no válido, DB2 devuelve SQLCODE –502. Si emite una sentencia UPDATE o DELETE WHERE CURRENT OF contra un cursor no válido, DB2 devuelve SQLCODE –910.

Dentro de los puntos de rescate, DB2 trata las tablas con la propiedad NOT LOGGED INITIALLY y las tablas temporales del siguiente modo:

Tablas NOT LOGGED INITIALLY
Si crea una tabla con la propiedad NOT LOGGED INITIALLY o modifica una tabla para que tenga la propiedad NOT LOGGED INITIALLY dentro de un punto de rescate, DB2 trata la sentencia ROLLBACK TO SAVEPOINT como una sentencia ROLLBACK WORK y retrotrae la unidad de trabajo completa.

DECLARE TEMPORARY TABLE dentro del punto de rescate
Si se declara una tabla temporal dentro de un punto de rescate, una sentencia ROLLBACK TO SAVEPOINT elimina la tabla temporal.

DECLARE TEMPORARY TABLE fuera del punto de rescate
Si se declara una tabla temporal fuera de un punto de rescate, una sentencia ROLLBACK TO SAVEPOINT no elimina la tabla temporal. Si la tabla temporal se declara como NOT LOGGED (si la cláusula NOT LOGGED no se utiliza en la sentencia DECLARE GLOBAL TEMPORARY TABLE), se retrotraen los cambios hechos en la tabla dentro del punto de rescate. Si la tabla temporal se declara como NOT LOGGED y se han cambiado los datos de la tabla dentro del punto de rescate, se suprimen todas las filas. En otro caso, si los datos no se han modificado, se conservan todas las filas.

Anidamiento de puntos de rescate

DB2® da soporte a los puntos de rescate anidados, es decir, el establecimiento de un punto de rescate dentro de otro punto de rescate. Puede definir un número

ilimitado de puntos de rescate y tampoco existe ninguna limitación respecto al número de niveles de puntos de rescate anidados.

Este ejemplo muestra el uso de puntos de rescate anidados:

```
CREATE TABLE Department (deptno CHAR(6), deptname VARCHAR(20), mgrno INT)
INSERT INTO Department VALUES ('A20', 'Marketing', 301)
SAVEPOINT savepoint1 ON ROLLBACK RETAIN® CURSORS
INSERT INTO Department VALUES ('B30', 'Finance', 520)
SAVEPOINT savepoint2 ON ROLLBACK RETAIN CURSORS
INSERT INTO Department VALUES ('C40', 'IT Support', 430)
SAVEPOINT savepoint3 ON ROLLBACK RETAIN CURSORS
INSERT INTO Department VALUES ('R50', 'Research', 150)
ROLLBACK TO SAVEPOINT savepoint3
ROLLBACK TO SAVEPOINT savepoint1
```

En este ejemplo existen dos niveles de anidamiento, donde el punto de rescate `savepoint3` está anidado dentro de `savepoint2` y `savepoint2` está anidado dentro de `savepoint1`. En `savepoint3`, una vez realizada la inserción del departamento `'R50'`, en este ejemplo se habrán añadido un total de cuatro filas a la tabla `Department`. Cuando se emite la sentencia ROLLBACK para `savepoint3`, solo se retrotrae la inserción realizada en `savepoint3`, dejando tres filas en la tabla `Department`. Cuando se emite la sentencia ROLLBACK para `savepoint1`, se retrotrae todo el trabajo realizado en `savepoint2` y `savepoint1`, dejando una sola fila en la tabla `Department`.

Puntos de rescate e inserciones en almacenamiento intermedio

Para mejorar el rendimiento de las aplicaciones DB2®, puede utilizar inserciones en almacenamiento intermedio en las aplicaciones, precompilando o vinculando con la opción INSERT BUF. Si la aplicación aprovecha tanto inserciones en almacenamiento como puntos de rescate, DB2 desecha el almacenamiento intermedio antes de ejecutar sentencias SAVEPOINT, RELEASE SAVEPOINT o ROLLBACK TO SAVEPOINT.

Conceptos relacionados:

- "Inserciones en almacenamiento intermedio en entornos de bases de datos particionadas" en la página 699

Información relacionada:

- "Mandato BIND" en la publicación *Consulta de mandatos*
- "Mandato PRECOMPILE" en la publicación *Consulta de mandatos*

Puntos de rescate y bloqueo de cursor

Si su aplicación utiliza puntos de rescate, puede evitar el bloqueo del cursor precompilando o vinculando la aplicación con la opción de precompilación BLOCKING NO. Aunque el bloqueo de cursores puede mejorar el rendimiento de la aplicación mediante la captación previa de varias filas, los datos devueltos por una aplicación que utiliza puntos de rescate y bloqueo de cursores puede que no reflejen los datos que se han confirmado en la base de datos.

Si no precompila la aplicación con la opción BLOCKING NO, y la aplicación emite una sentencia FETCH después de que se haya producido una operación ROLLBACK TO SAVEPOINT, es posible que la sentencia FETCH recupere datos

suprimidos. Por ejemplo, supongamos que la aplicación que contiene el siguiente SQL se precompila sin la opción BLOCKING NO:

```
CREATE TABLE t1(c1 INTEGER);
DECLARE CURSOR c1 AS 'SELECT c1 FROM t1 ORDER BY c1';
INSERT INTO t1 VALUES (1);
SAVEPOINT showFetchDelete;
   INSERT INTO t1 VALUES (2);
   INSERT INTO t1 VALUES (3);
   OPEN CURSOR c1;
   FETCH c1; --obtener primer valor y bloque de cursor
   ALTER TABLE t1... --añadir restricción
ROLLBACK TO SAVEPOINT;
FETCH c1; --recupera segundo valor de bloque de cursor
```

Cuando la aplicación emite la primera sentencia FETCH en la tabla "t1", el servidor DB2® envía un bloque de valores de columna (1, 2 y 3) a la aplicación cliente. El cliente almacena localmente estos valores de columna. Cuando la aplicación emite la sentencia ROLLBACK TO SAVEPOINT SQL, los valores de columna '2' y '3' se suprimen de la tabla. Después de la sentencia ROLLBACK TO SAVEPOINT, la siguiente sentencia FETCH de la tabla devuelve el valor de columna '2', aunque dicho valor ya no existe en la tabla. La aplicación recibe este valor porque aprovecha la opción de bloqueo de cursor para mejorar el rendimiento y accede a los datos que se han almacenado localmente.

Información relacionada:

- "Mandato BIND" en la publicación *Consulta de mandatos*
- "Mandato PRECOMPILE" en la publicación *Consulta de mandatos*

Puntos de rescate y gestores de transacciones que cumplen con XA

Si hay algún punto de rescate activo en una aplicación cuando un gestor de transacciones que cumple con XA emite una petición XA_END, DB2® emite una sentencia RELEASE SAVEPOINT.

Consideraciones sobre la programación de interfaces XA de X/Open

La Interfaz XA de es un estándar abierto para coordinar cambios en varios recursos, mientras que se asegura la integridad de dichos cambios. Normalmente, los productos de software conocidos como *supervisores del proceso de transacciones* utilizan la interfaz XA y, puesto que DB2 soporta esta interfaz, se puede acceder simultáneamente a una o más bases de datos DB2 como recursos en un entorno de este tipo.

DB2 requiere una consideración especial cuando funciona en un entorno de proceso de transacciones distribuidas (DTP) que utiliza la interfaz XA, puesto que se utiliza un modelo distinto para el proceso de transacciones, en comparación con las aplicaciones que se ejecutan de forma independiente de un supervisor de TP. Las características de este modelo de proceso de transacciones son:

- Dentro de una transacción se pueden modificar varios tipos de recursos recuperables (como, por ejemplo, bases de datos DB2).
- Los recursos se actualizan utilizando una confirmación en dos fases para cerciorarse de la integridad de las transacciones que se ejecutan.
- Los programas de aplicación envían peticiones para confirmar o retrotraer una transacción al producto supervisor de TP, en lugar de hacerlo a los gestores de

los recursos. Por ejemplo, en un entorno CICS®, una aplicación emitirá `EXEC CICS SYNCPOINT` para confirmar una transacción, y una emisión de `EXEC SQL COMMIT` a DB2 no sería válida y resultaría innecesaria.

- El supervisor de TP filtra la autorización para ejecutar transacciones y el software relacionado, por lo que los gestores de recursos tales como DB2 tratan el supervisor de TP como único usuario autorizado. Por ejemplo, cualquier uso de una transacción CICS debe ser autentificado por CICS, y se debe otorgar a CICS el privilegio de acceso a la base de datos, en lugar de al usuario final que invoca a la aplicación CICS.
- Normalmente, muchos programas (transacciones) se ponen en cola y se ejecutan en un servidor de bases de datos (que aparece ante DB2 como un solo programa de aplicación de ejecución prolongada).

Debido a la naturaleza exclusiva de este entorno, DB2 tiene un comportamiento y unos requisitos especiales para las aplicaciones codificadas para que se ejecuten en él:

- Dentro de una unidad de trabajo se puede conectar con varias bases de datos, y actualizarlas, sin tener en cuenta las opciones de precompilador o los valores del cliente de la unidad de trabajo distribuida.
- La sentencia DISCONNECT no está permitida y, de intentarla, se rechazará con SQLCODE -30090 (SQLSTATE 25000).
- La sentencia RELEASE no recibe soporte y se rechazará con -30090.
- Las sentencias COMMIT y ROLLBACK no están permitidas dentro de los procedimientos almacenados a los que accede una transacción de supervisor de TP.
- Cuando se inhabilitan explícitamente los flujos de confirmación en dos fases para una transacción (que se denomina transacción *LOCAL* en la terminología de la Interfaz XA), dentro de esta transacción sólo se puede acceder a una base de datos. Esta base de datos no puede ser una base de datos de sistema principal, AS/400® ni iSeries a la que se acceda utilizando la conectividad SNA. Se soportan transacciones locales para DB2® para OS/390® Versión 5 que utilizan la conectividad TCP/IP.
- Las transacciones LOCALES deben emitir SQL COMMIT o SQL ROLLBACK al final de cada transacción; de lo contrario, se considerará que la transacción forma parte de la próxima transacción que se procese.
- La conmutación entre conexiones de base de datos simultáneas se realiza mediante el uso de SQL CONNECT o de SQL SET CONNECTION. La autorización utilizada para una conexión no se puede cambiar especificando un ID de usuario o una contraseña en la sentencia CONNECT.
- Si un objeto de base de datos, tal como una tabla, una vista o un índice, no se califica completamente en una sentencia de SQL dinámico, se calificará implícitamente con el ID de autenticación simple bajo el que se ejecute el supervisor de TP, en lugar de hacerlo con el ID del usuario.
- Cualquier uso de las sentencias COMMIT o ROLLBACK de DB2 para transacciones que no sean LOCALES será rechazado. Se devolverán los códigos siguientes:
  - SQLCODE -925 (SQLSTATE 2D521) para una COMMIT estática
  - SQLCODE -926 (SQLSTATE 2D521) para una ROLLBACK estática
  - SQLCODE -426 (SQLSTATE 2D528) para una COMMIT dinámica
  - SQLCODE -427 (SQLSTATE 2D529) para una ROLLBACK dinámica
- Las peticiones de COMMIT o ROLLBACK por parte de la CLI también son rechazadas.
- Manejo de retrotracciones iniciadas por una base de datos:

En un entorno DTP, si un RM ha iniciado una retrotracción (por ejemplo, debido a un error del sistema o a una situación de punto muerto) para terminar su propia ramificación de una transacción global, no debe procesar ninguna otra petición del mismo proceso de aplicación hasta que se produzca una petición de punto de sincronismo iniciada por el gestor de transacciones. Esto incluye los puntos muertos que se producen dentro de un procedimiento almacenado. Para el gestor de bases de datos, esto significa que se rechacen todas las peticiones de SQL posteriores con SQLCODE -918 (SQLSTATE 51021), para informar al usuario de que debe retrotraer la transacción global con el servicio de puntos de sincronismo del gestor de transacciones, por ejemplo utilizando el mandato CICS SYNCPOINT ROLLBACK en un entorno CICS. En cambio si, por algún motivo, se solicita al TM que confirme la transacción, el RM informará al TM sobre la retrotracción y hará que el TM retrotraiga también los otros RM.

- Cursores declarados WITH HOLD:

  Los cursores declarados WITH HOLD se soportan en entornos XA/DTP para supervisores del proceso de transacciones CICS.

  En los casos en que no se soporten cursores declarados WITH HOLD, una sentencia OPEN será rechazada con SQLCODE -30090 (SQLSTATE 25000), código de razón 03.

  Es responsabilidad de las transacciones asegurarse de que los cursores especificados como WITH HOLD se cierren explícitamente cuando ya no sean necesarios; de lo contrario, es posible que otras transacciones los hereden, lo cual ocasionará conflictos o un uso innecesario de recursos.

  Si el supervisor de TP da soporte a cursores HOLD, `xa_commit`, `xa_rollback` y `xa_prepare` se deben emitir en la misma conexión que la transacción global.
- Las sentencias que actualizan o cambian una base de datos no están permitidas para las bases de datos que no soportan flujos de peticiones de confirmación en dos fases. Por ejemplo, el acceso a servidores de bases de datos de sistema principal, AS/400 o iSeries en entornos en los que el nivel 2 del protocolo DRDA® (DRDA2) no recibe soporte.
- Durante la ejecución, se puede determinar si una base de datos soporta actualizaciones en un entorno XA emitiendo una sentencia CONNECT. El tercer símbolo SQLERRD tendrá el valor 1 si la base de datos se puede actualizar; de lo contrario, tendrá el valor 2.
- Si están restringidas las actualizaciones, sólo se permitirán las sentencias de SQL siguientes:

```
CONNECT
DECLARE
DESCRIBE
EXECUTE IMMEDIATE (donde el primer símbolo o palabra clave es SET
                   pero no SET INTEGRITY)
OPEN CURSOR
FETCH CURSOR
CLOSE CURSOR
PREPARE (donde el primer símbolo o palabra clave que no sea un blanco
          ni un paréntesis de apertura es SET (que no sea SET INTEGRITY),
          SELECT, WITH o VALUES)
SELECT...INTO
VALUES...INTO
```

  Cualquier otro intento será rechazado con SQLCODE -30090 (SQLSTATE 25000).

  La sentencia PREPARE sólo se podrá utilizar para preparar sentencias SELECT. La sentencia EXECUTE IMMEDIATE también está permitida para ejecutar sentencias SET de SQL que no devuelvan ningún valor de salida, como por ejemplo la sentencia SET SQLID de DB2 Universal Database para z/OS y OS/390.

- Restricciones de las API:

  Las API que emiten internamente una confirmación en la base de datos y eluden el proceso de confirmación en dos fases serán rechazadas con SQLCODE -30090 (SQLSTATE 25000). Para ver una lista de estas API, consulte el apartado sobre restricciones en aplicaciones de actualización múltiple. Estas API no se soportan en una actualización múltiple (Connect Tipo 2).
- DB2 da soporte a un entorno XA/DTP de varias hebras.

Observe que las restricciones anteriores se aplican a las aplicaciones que se ejecutan en un entorno de supervisor de TP que utiliza la interfaz XA. Si las bases de datos DB2 no están definidas para que se utilicen con la interfaz XA, no se aplican estas restricciones; no obstante, sigue siendo necesario asegurarse de que las transacciones se codifiquen de manera que no dejen DB2 en un estado que afecte adversamente a la próxima transacción que se vaya a ejecutar.

Conceptos relacionados:

- "Security considerations for XA transaction managers" en la publicación *Administration Guide: Planning*
- "Configuration considerations for XA transaction managers" en la publicación *Administration Guide: Planning*
- "XA function supported by DB2 Universal Database" en la publicación *Administration Guide: Planning*
- "Actualización múltiple con DB2 Connect" en la página 749

Tareas relacionadas:

- "Updating host or iSeries database servers with an XA-compliant transaction manager" en la publicación *Administration Guide: Planning*

Enlace de aplicaciones y la interfaz XA de X/Open

Para producir una aplicación ejecutable, es necesario que enlace los objetos de la aplicación con las bibliotecas de lenguajes, las bibliotecas del sistema operativo, las bibliotecas normales del gestor de bases de datos y las bibliotecas del supervisor de TP y de los productos gestores de transacciones.

Gestión de transacciones MTS y COM+

Microsoft Transaction Server (MTS) y Microsoft Component Services (COM+) como gestor de transacciones

DB2® UDB se puede integrar por completo con Microsoft® Transaction Server (MTS) Versión 2.0 en Windows® NT o Microsoft Component Services (COM+) en Windows 2000 y Windows XP para coordinar la confirmación en dos fases con varios servidores de bases de datos DB2 UDB, zSeries® e iSeries™, así como con otros gestores de recursos compatibles con las especificaciones MTS o COM+.

Requisitos necesarios:

Para utilizar el soporte de transacciones distribuidas MTS o COM+, asegúrese de que se cumplen los siguientes requisitos para la máquina Windows en la que está instalado el cliente DB2:

- Windows NT® con MTS al nivel de Versión 2.0: Microsoft Hotfix 0772 o posterior

  MTS Versión 2.0 para Windows NT está disponible como parte de Windows NT 4.0 Option Pack. Puede bajar el Option Pack de:

  ```
  http://www.microsoft.com/ntserver/nts/downloads/recommended/NT4OptPk/
  ```

- Windows 2000: Service Pack 3 o posterior

Para aplicaciones CLI de DB2 que utilizan MTS o COM+:

- No cambie el valor por omisión del atributo de entorno SQL_ATTR_CONNECTION_POOLING CLI (el valor por omisión es SQL_CP_OFF)
- La instalación del controlador DB2 ODBC en sistemas operativos Windows añadirá automáticamente una nueva palabra clave al registro:

  ```
  HKEY_LOCAL_MACHINE\software\ODBC\odbcinit.ini\IBM DB2 ODBC Driver:
  Keyword Value Name: CPTimeout
  Data Type: REG_SZ
  Value: 60
  ```

Servidores de bases de datos DB2 soportados:

Los siguientes servidores reciben soporte para la actualización de varios sitios mediante transacciones coordinadas MTS o COM+:

- DB2 Universal Database™ Enterprise Server Edition (ESE)

  Nota: Las transacciones globales ligeramente agrupadas para MTS o COM+ no reciben soporte en entornos de proceso paralelo masivo (MPP). Las transacciones globales ligeramente agrupadas existen cuando cada uno de los procesos de aplicaciones accede a gestores de recursos como si fueran una transacción global separada; sin embargo, estos procesos de aplicaciones están bajo la coordinación del gestor de transacciones. Cada proceso de aplicación tendrá su propia ramificación de transacción dentro de un gestor de recursos. Cuando alguno de los procesos de aplicaciones, gestores de transacciones o gestores de recursos solicita una confirmación o retrotracción, las ramificaciones de la transacción finalizan simultáneamente. Es responsabilidad de la aplicación asegurar que no se produzca un punto muerto de recurso entre las ramificaciones.

  (Las transacciones globales estrechamente agrupadas existen cuando varios procesos de aplicaciones hacen turnos para hacer un trabajo bajo la misma ramificación de transacción en un gestor de recursos. Para el gestor de recursos, los dos procesos de aplicaciones son una sola entidad. El gestor de recursos debe asegurarse de que no se produzca un punto muerto dentro de la ramificación de la transacción.)
- DB2 Universal Database para z/OS™
- DB2 Universal Database para iSeries
- DB2 Server para VSE y VM

Consideraciones sobre la instalación y configuración:

A continuación se muestra un resumen de las consideraciones sobre la instalación y la configuración correspondientes a la utilización de MTS (COM+ se debe instalar por omisión como parte de Windows 2000):

- Instale MTS y el cliente DB2 en la misma máquina en la que se ejecuta la aplicación MTS.

- Si intervienen servidores de bases de datos iSeries o de sistema principal en una actualización de varios sitios:
  1. Instale la función DB2 Connect™ (DB2 Connect Enterprise Edition (EE) o DB2 UDB Enterprise Server Edition (ESE) con la función DB2 Connect instalada) en la máquina local o en una máquina remota. La función DB2 Connect permite que los servidores de bases de datos iSeries o de sistema principal participen en una transacción de actualización de varios sitios.
  2. Asegúrese de que el servidor DB2 Connect está habilitado para la actualización de varios sitios.

Conceptos relacionados:

- "X/Open distributed transaction processing model" en la publicación *Administration Guide: Planning*
- "Soporte de transacciones distribuidas MTS y COM+ e IBM OLE DB Provider" en la página 262
- "Tiempo de espera de transacciones de Microsoft Transaction Server (MTS) y Microsoft Component Services (COM+)" en la página 694
- "Soporte ligeramente agrupado con Microsoft Component Services (COM+)" en la página 693
- "Agrupación de conexiones ODBC y ADO con Microsoft Transaction Server (MTS) y Microsoft Component Services (COM+)" en la página 695

Tareas relacionadas:

- "Instalación de DB2 Connect Enterprise Edition (Windows)" en la publicación *Guía rápida de iniciación para DB2 Connect Enterprise Edition*

Información relacionada:

- "SQLSetConnectAttr function (CLI) - Set connection attributes" en la publicación *CLI Guide and Reference, Volume 2*
- "DB2 Connect product offerings" en la publicación *DB2 Connect User's Guide*
- "Connection attributes (CLI) list" en la publicación *CLI Guide and Reference, Volume 2*
- "Environment attributes (CLI) list" en la publicación *CLI Guide and Reference, Volume 2*

Soporte ligeramente agrupado con Microsoft Component Services (COM+)

Existen transacciones globales ligeramente agrupadas cuando cada uno de los procesos de aplicación accede a gestores de recursos como si estuviera en una transacción global separada; sin embargo, estos procesos de aplicación están bajo la coordinación del gestor de transacciones. Cada proceso de aplicación tendrá su propia ramificación de transacción dentro de un gestor de recursos. Cuando alguno de los procesos de aplicaciones, gestores de transacciones o gestores de recursos solicita una confirmación o retrotracción, las ramificaciones de la transacción finalizan simultáneamente. Es responsabilidad de la aplicación asegurar que no se produzca un punto muerto de recurso entre las ramificaciones.

DB2® Universal Database Versión 8 da soporte a transacciones globales ligeramente agrupadas para objetos COM+, sin tiempo de espera de bloqueo ni punto muerto, con las siguientes restricciones:

- El lenguaje de definición de datos (DDL) recibe soporte si se ejecuta en una sola ramificación mientras no hay ninguna otra transacción ligeramente agrupada activa. Si una ramificación ligeramente agrupada se intenta iniciar mientras una ramificación que ejecuta DDL está activa, la ramificación ligeramente agrupada se rechaza. Por el contrario, si hay al menos una transacción ligeramente agrupada activa, cualquier intento de ejecutar DDL en otra ramificación se rechaza.
- Las transacciones globales ligeramente agrupadas no reciben soporte en entornos de proceso paralelo masivo (MPP). En un entorno MPP, cada transacción global se trata de forma aislada, donde no se puede producir ningún punto muerto ni tiempo de espera excedido.
- El proceso de puntos de control y las sentencias de SQL se ejecutan en serie entre varias conexiones.
- Cuando se ha realizado una retrotracción implícita en una conexión, todas las ramificaciones de otras conexiones que están relacionadas con la transacción ligeramente agrupada devolverán el mensaje SQL0998N, con código de razón 225 y subcódigo 4: ″Sólo se permiten retrotracciones para esta transacción″.

Conceptos relacionados:

- "X/Open distributed transaction processing model" en la publicación *Administration Guide: Planning*
- "Soporte de transacciones distribuidas MTS y COM+ e IBM OLE DB Provider" en la página 262
- "Microsoft Transaction Server (MTS) y Microsoft Component Services (COM+) como gestor de transacciones" en la página 691
- "Tiempo de espera de transacciones de Microsoft Transaction Server (MTS) y Microsoft Component Services (COM+)" en la página 694
- "Agrupación de conexiones ODBC y ADO con Microsoft Transaction Server (MTS) y Microsoft Component Services (COM+)" en la página 695

Tiempo de espera de transacciones de Microsoft Transaction Server (MTS) y Microsoft Component Services (COM+)

El tiempo de espera de transacciones se puede establecer a través de las siguientes herramientas cuando se utiliza MTS o COM+:

- MTS (Microsoft Windows® NT): herramienta MTS Explorer
- COM+ (Microsoft Windows 2000 y XP): Servicios de componentes, situado bajo Herramientas administrativas del Panel de control de Windows

Si una transacción tarda más en ejecutarse que el valor de tiempo excedido de la transacción (el valor por omisión es 60 segundos), MTS o COM+ emitirá de forma asíncrona una terminación anormal para todos los gestores de recursos involucrados y la transacción completa terminará anormalmente.

La terminación anormal se convierte en una solicitud de retrotracción de DB2® en el servidor. La solicitud de retrotracción se serializa en la conexión en servidores que no sean DB2 para z/OS™ ni DB2 para iSeries™, para asegurar la integridad de los datos en el servidor de bases de datos. Cuando el servidor es DB2 para z/OS o DB2 para iSeries, la conexión se debe definir con la opción INTERRUPT_ENABLED en la entrada de catálogo de DCS para que, cuando se produzca un tiempo de espera excedido, la conexión procedente del servidor DB2 Connect™ con el servidor z/OS o iSeries se desconecte, forzando una retrotracción en el servidor z/OS o iSeries.

Como resultado:

- Si la conexión está desocupada, la retrotracción se ejecuta de inmediato.
- Si se está procesando una sentencia de SQL de larga ejecución, la solicitud de retrotracción espera hasta que finalice la sentencia de SQL.

Conceptos relacionados:

- "X/Open distributed transaction processing model" en la publicación *Administration Guide: Planning*
- "DCS directory values" en la publicación *DB2 Connect User's Guide*
- "Soporte de transacciones distribuidas MTS y COM+ e IBM OLE DB Provider" en la página 262
- "Proceso de peticiones de interrupción" en la página 743
- "Microsoft Transaction Server (MTS) y Microsoft Component Services (COM+) como gestor de transacciones" en la página 691
- "Soporte ligeramente agrupado con Microsoft Component Services (COM+)" en la página 693
- "Agrupación de conexiones ODBC y ADO con Microsoft Transaction Server (MTS) y Microsoft Component Services (COM+)" en la página 695

Agrupación de conexiones ODBC y ADO con Microsoft Transaction Server (MTS) y Microsoft Component Services (COM+)

La agrupación de conexiones permite a una aplicación utilizar una conexión de una agrupación de conexiones, de modo que la conexión no se tiene que restablecer cada vez que se desea utilizar. Cuando una conexión se crea y se coloca en una agrupación, una aplicación puede reutilizar dicha conexión sin tener que realizar el proceso completo de conexión. La conexión se agrupa cuando la aplicación se desconecta de la fuente de datos y se otorga a una nueva conexión cuyos atributos son los mismos.

Agrupación de conexiones ODBC:

La agrupación de conexiones es una característica de ODBC Driver Manager desde ODBC 2.x. Con la última versión de ODBC Driver Manager (versión 3.5) disponible como parte de la descarga de Microsoft® Data Access Components (MDAC), la agrupación de conexiones tiene algunos cambios de configuración y un nuevo comportamiento para las conexiones ODBC de objetos MTS COM+ transaccionales.

ODBC Driver Manager 3.5 necesita que el controlador ODBC registre una nueva palabra clave en el registro para permitir que se active la agrupación de conexiones. La palabra clave es:

```
Key Name: SOFTWARE\ODBC\ODBCINST.INI\IBM DB2® ODBC DRIVER
Name: CPTimeout
Type: REG_SZ
Data: 60
```

El controlador DB2 ODBC correspondiente al sistema operativo Windows® da soporte completo a la agrupación de conexiones; por lo tanto, esta palabra clave se registra.

El valor por omisión, 60, significa que la conexión se agrupará durante 60 segundos antes de que se desconecte.

En un entorno con mucha actividad, es mejor aumentar el valor CPTimeout a un número mayor pare evitar demasiadas conexiones y desconexiones físicas, puesto que estas consumen gran cantidad de recursos del sistema, incluidos recursos de memoria del sistema y de pila de comunicaciones.

Además, para asegurar que se utiliza la misma conexión entre objetos de la misma transacción en una máquina de varios procesadores, debe desactivar el soporte de "agrupación múltiple por procesador". Para ello, copie el siguiente valor de registro en un archivo denominado `odbcpool.reg`, guárdelo como un archivo de texto plano y emita el mandato **odbcpool.reg**. El sistema operativo Windows importará este valor de registro.

```
REGEDIT4

[HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI\ODBC Connection Pooling]
"NumberOfPools"="1"
```

Sin esta palabra clave establecida en 1, MTS o COM+ pueden agrupar conexiones correspondiente a la misma transacción en distintas agrupaciones, por lo que pueden no reutilizar la misma conexión.

Agrupación de conexiones ADO:

Si los objetos MTS o COM+ utilizan ADO para acceder a la base de datos, debe desactivar la agrupación de recursos OLE DB de modo que el proveedor Microsoft OLE DB para ODBC (MSDASQL) no interfiera con la agrupación de conexiones ODBC. Esta característica se inicializa en `OFF` en ADO 2.0, pero se inicializa en `ON` en ADO 2.1. Para desactivar la agrupación de recursos OLE DB, copie las siguientes líneas en un archivo denominado `oledb.reg`, guárdelo como un archivo de texto plano y emita el mandato **oledb.reg**. El sistema operativo Windows importará estos valores de registro.

```
REGEDIT4

[HKEY_CLASSES_ROOT\CLSID\{c8b522cb-5cf3-11ce-ade5-00aa0044773d}]
@="MSDASQL"
"OLEDB_SERVICES"=dword:fffffffc
```

Conceptos relacionados:

- "X/Open distributed transaction processing model" en la publicación *Administration Guide: Planning*
- "Soporte de transacciones distribuidas MTS y COM+ e IBM OLE DB Provider" en la página 262
- "Microsoft Transaction Server (MTS) y Microsoft Component Services (COM+) como gestor de transacciones" en la página 691
- "Tiempo de espera de transacciones de Microsoft Transaction Server (MTS) y Microsoft Component Services (COM+)" en la página 694
- "Soporte ligeramente agrupado con Microsoft Component Services (COM+)" en la página 693

Capítulo 31. Consideraciones sobre programación para entornos de bases de datos particionadas

Cursores FOR READ ONLY en un entorno de bases de datos particionadas 697
DDS dirigida y elusión local 697
DDS dirigida y elusión local en entornos de bases de datos particionadas 697
DDS dirigida en entornos de bases de datos particionadas 698
Elusión local en entornos de bases de datos particionadas 698
Inserciones colocadas en almacenamiento intermedio 699
Inserciones en almacenamiento intermedio en entornos de bases de datos particionadas . . . 699
Consideraciones sobre la utilización de inserciones en almacenamiento intermedio . . 702
Restricciones en la utilización de inserciones en almacenamiento intermedio 704
Ejemplo de extracción un gran volumen de datos en una base de datos particionada 705
Creación de un entorno simulado de bases de datos particionadas 710
Resolución de problemas 710
Consideraciones sobre el manejo de errores en entornos de bases de datos particionadas . . . 710
Errores graves en entornos de bases de datos particionadas 711
Varias estructuras SQLCA fusionadas 711
Partición que devuelve el error 712
Aplicaciones en bucle o suspendidas 712

Cursores FOR READ ONLY en un entorno de bases de datos particionadas

Si declara un cursor desde el cual sólo pretende leer, incluya FOR READ ONLY o FOR FETCH ONLY en la declaración del OPEN CURSOR. (FOR READ ONLY y FOR FETCH ONLY son sentencias equivalentes.) Los cursores FOR READ ONLY permiten que la partición coordinadora recupere varias filas a la vez, mejorando en gran medida el rendimiento de las sentencias FETCH posteriores. Cuando los cursores no se declaran explícitamente como FOR READ ONLY, la partición coordinadora los trata como cursores actualizables. Los cursores actualizables incurren en un gasto considerable, puesto que requieren que la partición coordinadora sólo recupere una única fila por cada FETCH.

DDS dirigida y elusión local

Las secciones siguientes describen las consideraciones a tener en cuenta para utilizar DDS dirigida y elusión local en entornos de bases de datos particionadas.

DDS dirigida y elusión local en entornos de bases de datos particionadas

Para optimizar las aplicaciones de proceso de transacciones en línea (OLTP), es posible que desee evitar las sentencias simples de SQL que requieran un proceso en todas las particiones de bases de datos. Debe diseñar la aplicación de forma que las sentencias de SQL puedan recuperar datos de particiones de bases de datos únicas. Las técnicas de subsección distribuida dirigida (DSS) y de elusión local evitan el gasto en que incurre la partición coordinadora al comunicar con una o con todas las particiones asociadas.

Conceptos relacionados:

- "DDS dirigida en entornos de bases de datos particionadas" en la página 698
- "Elusión local en entornos de bases de datos particionadas" en la página 698

DDS dirigida en entornos de bases de datos particionadas

Una subsección distribuida (DSS) es la acción de enviar subsecciones a la partición de base de datos que necesita realizar trabajo para una consulta paralela. También describe la iniciación de subsecciones mediante la invocación de valores específicos, como por ejemplo valores de variables en un entorno OLTP. Una *DDS dirigida* utiliza la clave de particionamiento de tabla para dirigir una consulta a una sola partición. Utilice este tipo de consulta en la aplicación para evitar la actividad general de la partición coordinadora necesaria para difundir una consulta a todas las particiones.

A continuación mostramos un fragmento de una sentencia SELECT de ejemplo que se puede aprovechar de las DSS dirigidas:

```
SELECT ... FROM t1
  WHERE PARTKEY=:hostvar
```

Cuando la partición coordinadora recibe la consulta, determina qué partición de base de datos contiene el subconjunto de datos para *:hostvar* y dirige la consulta, específicamente, a dicha partición de base de datos.

Para optimizar la aplicación utilizando DSS dirigidas, divida las consultas complejas en varias consultas simples. Por ejemplo, en la consulta siguiente la partición coordinadora compara la clave de particionamiento con varios valores. Dado que los datos que se ajustan a la consulta residen en varias particiones de base de datos, la partición coordinadora difunde la consulta a todas las particiones de base de datos:

```
SELECT ... FROM t1
  WHERE PARTKEY IN (:hostvar1, :hostvar2)
```

En lugar de esto, divida la consulta en varias sentencias SELECT (cada una de ellas con una sola variable del lenguaje principal) o utilice una única sentencia SELECT con UNION para lograr el mismo resultado. La partición coordinadora se puede aprovechar de sentencias SELECT más sencillas para utilizar DSS dirigidas a fin de comunicar únicamente con las particiones de base de datos necesarias. La consulta optimizada tiene el aspecto siguiente:

```
SELECT ... AS res1 FROM t1
  WHERE PARTKEY=:hostvar1
  UNION
SELECT ... AS res2 FROM t1
  WHERE PARTKEY=:hostvar2
```

Observe que la técnica anterior sólo mejorará el rendimiento si el número de selecciones de UNION es significativamente menor que el número de particiones.

Elusión local en entornos de bases de datos particionadas

Una forma especializada de la consulta DSS dirigida accede a los datos almacenados únicamente en la partición coordinadora. Esto se denomina *elusión local* porque la partición coordinadora completa la consulta sin tener que comunicar con otra partición.

Si es posible, la elusión local se habilita automáticamente, pero el usuario puede incrementar su uso dirigiendo transacciones a la partición de base de datos que contiene los datos correspondientes a dicha transacción. Una técnica para hacerlo consiste en hacer que un cliente remoto mantenga conexiones con cada una de las particiones de bases de datos. Luego una transacción puede utilizar la conexión

correcta según la clave de particionamiento de entrada. Otra técnica consiste en agrupar las transacciones por particiones de bases de datos y disponer de un servidor de aplicaciones separado para cada partición de base de datos.

Para determinar el número de la partición de base de datos en la que residen los datos de la transacción, puede utilizar la API `sqlugrpn` (Obtener número de particionamiento de filas). Esta API permite que una aplicación calcule con eficacia el número de partición de una fila, dada la clave de particionamiento.

Otra alternativa consiste en utilizar el programa de utilidad `db2atld` para dividir los datos de entrada por número de partición y ejecutar una copia de la aplicación en cada partición de base de datos.

Información relacionada:

- "sqlugrpn - Get Row Partitioning Number" en la publicación *Administrative API Reference*
- "db2atld - Mandato Cargador automático" en la publicación *Consulta de mandatos*

Inserciones colocadas en almacenamiento intermedio

Las secciones siguientes describen las consideraciones a tener en cuenta para utilizar inserciones colocadas en almacenamiento intermedio en entornos de bases de datos particionadas.

Inserciones en almacenamiento intermedio en entornos de bases de datos particionadas

Una inserción en almacenamiento intermedio es una sentencia de inserción que se aprovecha de las colas de tablas para poner en el almacenamiento intermedio las filas que se insertan, obteniendo así una mejora significativa en el rendimiento. Para utilizar una inserción en almacenamiento intermedio, una aplicación se debe preparar o vincular con la opción INSERT BUF.

Las inserciones en almacenamiento intermedio pueden tener como consecuencia una mejora sustancial en el rendimiento de las aplicaciones que realizan inserciones. Normalmente, se puede utilizar una inserción en almacenamiento intermedio en las aplicaciones en que se utiliza una sola sentencia de inserción (y ninguna otra sentencia de modificación de bases de datos) dentro de un bucle para insertar muchas filas, y si el origen de los datos es una cláusula VALUES de la sentencia INSERT. Habitualmente, la sentencia INSERT hace referencia a una o más variables del lenguaje principal que cambian de valor durante las sucesivas ejecuciones del bucle. La cláusula VALUES puede especificar una sola fila o varias filas.

Las aplicaciones típicas de soporte de decisiones requieren una carga e inserción periódica de datos nuevos. Estos datos pueden representar cientos de miles de filas. Puede preparar y vincular las aplicaciones de forma que utilicen inserciones en almacenamiento intermedio cuando carguen tablas.

Para hacer que una aplicación utilice inserciones en almacenamiento intermedio, use el mandato PREP para procesar el archivo fuente del programa de aplicación, o utilice el mandato BIND en el archivo de vinculación resultante. En ambas situaciones, se debe especificar la opción INSERT BUF.

Nota: Las inserciones en almacenamiento intermedio ocasionan que se lleven a cabo los pasos siguientes:

1. El gestor de bases de datos abre un almacenamiento intermedio de 4 KB para cada partición de base de datos en que reside la tabla.
2. La sentencia INSERT con la cláusula VALUES emitida por la aplicación hace que la o las fila(s) se coloque(n) en el o los almacenamiento(s) intermedio(s) apropiado(s).
3. El gestor de bases de datos devuelve el control a la aplicación.
4. Las filas del almacenamiento intermedio se envían a la partición cuando se llena el almacenamiento intermedio, o cuando se produce un suceso que hace que se envíen las filas de un almacenamiento intermedio parcialmente lleno. Un almacenamiento parcialmente lleno se vacía cuando se da una de las situaciones siguientes:
   - La aplicación emite una sentencia COMMIT (implícita o explícitamente a través de la terminación de la aplicación) o ROLLBACK.
   - La aplicación emite otra sentencia que hace que se tome un punto de rescate. Las sentencias de cursor OPEN, FETCH y CLOSE no ocasionan que se tome un punto de rescate ni cierran una inserción en almacenamiento intermedio abierta.

     Las sentencias de SQL siguientes cerrarán una inserción en almacenamiento intermedio abierta:
     - BEGIN COMPOUND SQL
     - COMMIT
     - DDL
     - DELETE
     - END COMPOUND SQL
     - EXECUTE IMMEDIATE
     - GRANT
     - INSERT en otra tabla
     - OPEN CURSOR para una sentencia de selección completa o cambio de datos
     - PREPARE de la misma sentencia dinámica (por nombres) realizando inserciones en almacenamiento intermedio
     - REDISTRIBUTE DATABASE PARTITION GROUP
     - RELEASE SAVEPOINT
     - REORG
     - REVOKE
     - ROLLBACK
     - ROLLBACK TO SAVEPOINT
     - RUNSTATS
     - SAVEPOINT
     - SELECT INTO
     - UPDATE
     - Ejecución de cualquier otra sentencia, pero no otra ejecución (bucle) de la INSERT en almacenamiento intermedio
     - Fin de la aplicación

     Las API siguientes cerrarán una inserción en almacenamiento intermedio abierta:
     - BIND (API)
     - REBIND (API)
     - RUNSTATS (API)
     - REORG (API)
     - REDISTRIBUTE (API)

En cualquiera de estas situaciones en que otra sentencia cierra la inserción en almacenamiento intermedio, la partición coordinadora espera hasta que todas las particiones de bases de datos reciben los almacenamientos intermedios y se insertan las filas. A continuación, ejecuta la otra sentencia (aquélla que cierra la inserción en almacenamiento intermedio), siempre que todas las filas se hayan insertado satisfactoriamente.

La interfaz estándar en un entorno particionado (sin inserción en almacenamiento intermedio) carga una fila cada vez, llevando a cabo los pasos siguientes (suponiendo que la aplicación se ejecuta localmente en una de las particiones de bases de datos):

1. La partición coordinadora pasa la fila al gestor de bases de datos que se encuentra en la misma partición.
2. El gestor de bases de datos utiliza un procedimiento de hash indirecto para determinar la partición de base de datos en que se debe colocar la fila:
   - La partición de destino recibe la fila.
   - La partición de destino inserta la fila localmente.
   - La partición de destino envía una respuesta a la partición coordinadora.
3. La partición coordinadora recibe la respuesta de la partición de destino.
4. La partición coordinadora pasa la respuesta a la aplicación.

   No se confirma la inserción hasta que la aplicación emite una sentencia COMMIT.
5. Cualquier sentencia INSERT que contenga una cláusula VALUES es candidata a una inserción en almacenamiento intermedio, independientemente del número de filas o del tipo de elementos contenidos en las filas. Es decir, los elementos pueden ser constantes, registros especiales, variables del lenguaje principal, expresiones, funciones, etc.

Para una sentencia INSERT determinada con la cláusula VALUES, es posible que el compilador SQL de DB2® no coloque en el almacenamiento intermedio la inserción en base a consideraciones sobre la semántica, el rendimiento o la implementación. Si prepara o vincula su aplicación con la opción INSERT BUF, asegúrese de que no depende de una inserción en almacenamiento intermedio. Esto significa que:

- Se puede informar de errores de forma asíncrona para las inserciones en almacenamiento intermedio, o de forma síncrona para las inserciones normales. Si se informa de forma asíncrona, un error de inserción se puede notificar en una inserción posterior en el almacenamiento intermedio, o en la *otra* sentencia que cierre el almacenamiento intermedio. La sentencia que informa del error no se ejecuta. Por ejemplo, piense en la utilización de una sentencia COMMIT para cerrar un bucle de inserciones en almacenamiento intermedio. La confirmación informa de SQLCODE -803 (SQLSTATE 23505) debido a una clave duplicada procedente de una inserción anterior. En este escenario, no se ejecuta la confirmación. Si desea que la aplicación efectúe realmente la confirmación, por ejemplo, algunas actualizaciones realizadas antes de entrar en el bucle de inserciones en almacenamiento intermedio, debe volver a emitir la sentencia COMMIT.
- Las filas insertadas pueden resultar visibles inmediatamente mediante una sentencia SELECT que utilice un cursor sin inserción en almacenamiento intermedio. Con una inserción en almacenamiento intermedio, las filas no serán visibles de inmediato. Si precompila o vincula la aplicación con la opción INSERT BUF, no la escriba de forma que dependa de estas filas seleccionadas por el cursor.

Las inserciones en almacenamiento intermedio tienen como consecuencia las ventajas siguientes:

- Sólo se envía un mensaje desde la partición de destino a la partición coordinadora por cada almacenamiento intermedio recibido por la partición de destino.
- Un almacenamiento intermedio puede contener un elevado número de filas, especialmente si las filas son pequeñas.
- Se produce un proceso paralelo a medida que se realizan inserciones en las particiones, mientras la partición coordinadora está recibiendo filas nuevas.

Una aplicación que se vincula con INSERT BUF se debe escribir de forma que la misma sentencia INSERT con la cláusula VALUES se reitere repetidamente antes de que se emita cualquier sentencia o API que cierre una inserción en almacenamiento intermedio.

Nota: Para impedir que las inserciones en almacenamiento intermedio llenen el registro de transacciones, debe realizar confirmaciones periódicas.

Conceptos relacionados:

- "Creación y preparación del archivo fuente" en la página 63
- "Creación de paquetes mediante el mandato BIND" en la página 71
- "Consideraciones sobre la utilización de inserciones en almacenamiento intermedio" en la página 702
- "Restricciones en la utilización de inserciones en almacenamiento intermedio" en la página 704

Consideraciones sobre la utilización de inserciones en almacenamiento intermedio

Las inserciones en almacenamiento intermedio presentan comportamientos que pueden afectar a un programa de aplicación. Este comportamiento es debido a la naturaleza asíncrona de las inserciones en almacenamiento intermedio. En base a los valores de la clave de particionamiento de la fila, cada fila insertada se coloca en un almacenamiento intermedio destinado a la partición correcta. Estos almacenamientos intermedios se envían a sus particiones de destino cuando se llenan o cuando un suceso ocasiona que se vacíen. Cuando diseñe y codifique la aplicación, debe tener en cuenta los aspectos siguientes:

- Existen determinadas condiciones de error de las que no se informa cuando se ejecuta la sentencia INSERT. Se informa de ellas más tarde, cuando se ejecuta la primera sentencia que no sea INSERT (o INSERT en otro tabla), como por ejemplo DELETE, UPDATE, COMMIT o ROLLBACK. Cualquier sentencia o API que cierre la sentencia de inserción en almacenamiento intermedio puede ver el informe de errores. Asimismo, cualquier invocación de la propia inserción puede ver un error de una fila insertada previamente. Por otra parte, si otra sentencia informa de un error de una inserción en almacenamiento intermedio, como por ejemplo UPDATE o COMMIT, DB2® no intentará ejecutar dicha sentencia.
- Un error detectado durante la inserción de un *grupo de filas* hace que se retrotraigan todas las filas de dicho grupo. Un grupo de filas se define como todas las filas insertadas mediante ejecuciones de una sentencia de inserción en almacenamiento intermedio:
  - Desde el principio de la unidad de trabajo,
  - Desde que se preparó la sentencia (si ésta es dinámica), o

  - Desde la ejecución anterior de otra sentencia de actualización. Para ver una lista de las sentencias que cierran (o vacían) una inserción en almacenamiento intermedio, consulte la descripción de inserciones en almacenamiento intermedio en entornos de bases de datos particionadas.
- Es posible que una fila insertada no resulte visible inmediatamente para la sentencias SELECT emitidas después de INSERT por el mismo programa de aplicación, en caso de que SELECT se ejecute utilizando un cursor.

Una sentencia INSERT en almacenamiento intermedio puede estar abierta o cerrada. La primera invocación de la sentencia abre la INSERT en almacenamiento intermedio, se añade la fila al almacenamiento intermedio apropiado y se devuelve el control a la aplicación. Las invocaciones posteriores añaden filas al almacenamiento intermedio, dejando la sentencia abierta. Mientras la sentencia está abierta, los almacenamientos intermedios se pueden enviar a sus particiones de destino, donde se insertan las filas en la partición de la tabla de destino. Si se invoca a cualquier sentencia o API que cierra una inserción en almacenamiento intermedio mientras hay una sentencia INSERT en almacenamiento intermedio abierta (incluida una invocación de *otra* sentencia INSERT en almacenamiento intermedio), o si se emite una sentencia PREPARE para una sentencia INSERT en almacenamiento intermedio abierta, antes de procesar la nueva petición se cierra la sentencia abierta. Si se cierra la sentencia INSERT en almacenamiento intermedio, se vacían los almacenamientos intermedios restantes. Entonces se envían las filas a las particiones de destino y se insertan. El proceso de la nueva petición sólo comienza después de que se envíen todos los almacenamientos intermedios y se inserten todas las filas.

Si se detectan errores durante el cierre de la sentencia INSERT, la SQLCA de la nueva petición se llenará con la descripción del error y no se ejecutará la petición. Asimismo, se eliminará de la base de datos el grupo entero de filas que se hubieran insertado mediante la sentencia INSERT en almacenamiento intermedio *desde que se abrió*. El estado de la aplicación será tal como se haya definido para el error concreto que se haya detectado. Por ejemplo:

- Si el error es un punto muerto, se retrotrae la transacción (incluidos los cambios efectuados antes de que se abriera la sección de inserción en almacenamiento intermedio).
- Si el error es una violación de una clave exclusiva, el estado de la base de datos es igual que antes de que se abriera la sentencia. La transacción permanece activa y los cambios efectuados antes de que se abriera la sentencia no se ven afectados.

Por ejemplo, considere la aplicación siguiente que está vinculada con la opción de inserción en almacenamiento intermedio:

```
   EXEC SQL UPDATE t1 SET COMMENT='about to start inserts';
DO UNTIL EOF OR SQLCODE < 0;
  READ VALUE OF hv1 FROM A FILE;
  EXEC SQL INSERT INTO t2 VALUES (:hv1);
  IF 1000 INSERTS DONE, THEN DO
     EXEC SQL INSERT INTO t3 VALUES ('another 1000 done');
     RESET COUNTER;
  END;
END;
EXEC SQL COMMIT;
```

Suponga que el archivo contiene 8 000 valores, pero el valor 3 258 no es lícito (por ejemplo, contiene una violación de la clave exclusiva). Cada 1 000 inserciones dan como resultado la ejecución de otra sentencia de SQL que, a continuación, cierra la sentencia `INSERT INTO t2`. Durante el cuarto grupo de 1 000 inserciones, se

detectará el error del valor 3 258. Se puede detectar después de la inserción de más valores (y no necesariamente después del siguiente). En esta situación, se devuelve un código de error para la sentencia `INSERT INTO t2`.

También es posible que se detecte el error cuando se intente una inserción en la tabla `t3`, la cual cerrará la sentencia `INSERT INTO t2`. En esta situación, se devuelve el código de error para la sentencia `INSERT INTO t3`, aunque el error corresponda a la tabla `t2`.

En cambio, suponga que tiene que insertar 3 900 filas. Antes de que se le indique el error de la fila número 3 258, la aplicación puede salir del bucle e intentar emitir una sentencia COMMIT. El código de retorno de la violación de la clave exclusiva se emitirá para la sentencia COMMIT y no se ejecutará COMMIT. Si la aplicación desea confirmar (COMMIT) las 3000 filas que se encuentran hasta ahora en la base de datos (la última ejecución de `EXEC SQL INSERT INTO t3 ...` finaliza el punto de rescate para estas 3000 filas), habrá que *volver a emitir* la sentencia COMMIT. También se aplican consideraciones parecidas a ROLLBACK.

Nota: Cuando utilice inserciones en almacenamiento intermedio, debe supervisar cuidadosamente los SQLCODE devueltos para evitar que una tabla quede en un estado indeterminado. Por ejemplo, si eliminara la cláusula `SQLCODE < 0` de la sentencia THEN DO del ejemplo anterior, la tabla podría terminar por contener un número indeterminado de filas.

Conceptos relacionados:

- "Inserciones en almacenamiento intermedio en entornos de bases de datos particionadas" en la página 699

Restricciones en la utilización de inserciones en almacenamiento intermedio

Se aplican las restricciones siguientes a las inserciones en almacenamiento intermedio:

- Para que una aplicación se aproveche de las inserciones en almacenamiento intermedio, se tiene que cumplir una de las condiciones siguientes:
  - La aplicación debe estar preparada mediante PREP o vinculada con el mandato BIND y se debe especificar la opción INSERT BUF.
  - La aplicación debe estar vinculada utilizando las API BIND o PREP con la opción SQL_INSERT_BUF.
- Si la sentencia INSERT con la cláusula VALUES incluye campos largos o LOB en la lista de columnas explícita o implícita, se pasa por alto la opción INSERT BUF para esta sentencia y se realiza una inserción normal, no en el almacenamiento intermedio. Ésta no es una condición de error y no se emite ningún mensaje de error ni de aviso.
- Una INSERT con una selección completa no se ve afectada por INSERT BUF. Una inserción en almacenamiento intermedio no mejora el rendimiento de este tipo de INSERT.
- Las inserciones en almacenamiento intermedio sólo se pueden utilizar en aplicaciones y no mediante inserciones emitidas por el CLP, puesto que éstas se realizan a través de la sentencia EXECUTE IMMEDIATE.

Luego, la aplicación se puede ejecutar desde cualquier plataforma cliente soportada.

Ejemplo de extracción un gran volumen de datos en una base de datos particionada

Aunque DB2 Universal Database proporciona características excelentes para el proceso de consultas paralelas, el único punto de conexión de una aplicación o un mandato EXPORT puede convertirse en un embotellamiento si se extraen grandes volúmenes de datos. Este cuello de botella se produce porque el proceso de pasar los datos del gestor de bases de datos a la aplicación hace un gran uso de la CPU que se ejecuta en una sola partición (normalmente, también en un solo procesador).

DB2 Universal Database brinda varios métodos para vencer el embotellamiento, de forma que el volumen de datos extraídos se escala linealmente por unidad de tiempo con un número de procesadores incrementado. El ejemplo siguiente describe la idea básica que se encuentra detrás de estos métodos.

Suponga que tiene una tabla llamada EMPLOYEE que está almacenada en 20 particiones de bases de datos, y que genera una lista de correo (FIRSTNME, LASTNAME, JOB) de todos los empleados que pertenecen a un departamento legítimo (es decir, en que WORKDEPT no es NULL).

La consulta siguiente se ejecuta en cada partición y luego genera el conjunto completo de respuestas en una sola partición (la partición coordinadora):

```
SELECT FIRSTNME, LASTNAME, JOB FROM EMPLOYEE WHERE WORKDEPT IS NOT NULL
```

Pero la consulta siguiente se puede ejecutar en cada partición correspondiente a la base de datos (es decir, si existen cinco particiones, se necesitan cinco consultas separadas, una en cada partición). Cada consulta genera el conjunto de todos los nombres de empleado cuyo registro se encuentra en la partición concreta en que se ejecuta la consulta. Cada conjunto de resultados local se puede redirigir a un archivo. Luego es necesario fusionar los conjuntos de resultados en un solo conjunto.

En AIX®, puede utilizar una propiedad de NFS (Network File System) para automatizar la fusión. Si todas las particiones dirigen sus conjuntos de respuestas al mismo archivo de un montaje por NFS, los resultados se fusionan. Observe que una utilización de NFS sin bloquear la respuesta en almacenamientos intermedios grandes tiene como consecuencia un rendimiento muy pobre.

```
SELECT FIRSTNME, LASTNAME, JOB FROM EMPLOYEE WHERE WORKDEPT IS NOT NULL
              AND NODENUMBER(NAME) = CURRENT NODE
```

El resultado se puede almacenar en un archivo local (lo que significa que el resultado final constará de 20 archivos, cada uno de los cuales contendrá una porción del conjunto de respuestas completo), o en un solo archivo montado por NFS.

En el ejemplo siguiente se utiliza el segundo método, de forma que el resultado se encuentra en un único archivo montado por NFS para los 20 nodos. El mecanismo de bloqueo de NFS asegura la colocación en serie de las grabaciones en el archivo de resultados por parte de las distintas particiones. Observe que este ejemplo, tal como se presenta, se ejecuta en la plataforma AIX con un sistema de archivos NFS instalado.

```
#define _POSIX_SOURCE
#define INCL_32

#include <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sqlenv.h>
#include <errno.h>
#include <sys/access.h>
#include <sys/flock.h>
#include <unistd.h>

#define BUF_SIZE 1500000  /* Almacenamiento intermedio local para almacenar
                             los registros captados */
#define MAX_RECORD_SIZE 80 /* >= tamaño de un registro grabado */

int main(int argc, char *argv[]) {

    EXEC SQL INCLUDE SQLCA;
    EXEC SQL BEGIN DECLARE SECTION;
       char dbname[10];  /* Nombre de base de datos (argumento
                            del programa) */
       char userid[9];
       char passwd[19];
       char first_name[21];
       char last_name[21];
       char job_code[11];
    EXEC SQL END DECLARE SECTION;

       struct flock unlock ;  /* estructuras y variables para manejar */
       struct flock lock ;  /* el mecanismo de bloqueo NFS */
       int lock_command ;
       int lock_rc ;
       int iFileHandle ;  /* archivo de salida */
       int iOpenOptions = 0 ;
       int iPermissions ;
       char * file_buf ;  /* puntero al almacenamiento intermedio
                             en que se acumulan los registros captados */
       char * write_ptr ;  /* posición en que se graba el siguiente
                              registro */
       int buffer_len = 0 ;  /* longitud de la porción utilizada del
                                almacenamiento intermedio */

    /* Inicialización */

       lock.l_type = F_WRLCK;  /* Petición de bloqueo exclusivo de grabación */
       lock.l_start = 0;  /* Para bloquear el archivo entero */
       lock.l_whence = SEEK_SET;
       lock.l_len = 0;
       unlock.l_type = F_UNLCK;  /* Petición de liberación de un bloqueo */
       unlock.l_start = 0;  /* Para desbloquear el archivo entero */
       unlock.l_whence = SEEK_SET;
       unlock.l_len = 0;
       lock_command = F_SETLKW;  /* Establecer el bloqueo */
       iOpenOptions = O_CREAT;  /* Crear el archivo si no existe */
       iOpenOptions |= O_WRONLY;  /* Abrir sólo para grabación */

    /* Conectar con la base de datos */

       if (argc == 3) {
          strcpy( dbname, argv[2] ); /* obtener nombre de base de datos
                                        del argumento */
          EXEC SQL CONNECT TO :dbname IN SHARE MODE ;
    if ( SQLCODE != 0 ) {
             printf( "Error: CONNECT TO the database failed. SQLCODE = %ld\n",
             SQLCODE );
          exit(1);
      }
          }
          else if ( argc == 5 ) {
```

```
        strcpy( dbname, argv[2] ); /* obtener nombre de base de datos
                                      del argumento */
strcpy (userid, argv[3]);
strcpy (passwd, argv[4]);
EXEC SQL CONNECT TO :dbname IN SHARE MODE USER :userid USING :passwd;
if ( SQLCODE != 0 ) {
   printf( "Error: CONNECT TO the database failed. SQLCODE = %ld\n",
           SQLCODE );
     exit( 1 );
       }
     }
                 else
  printf ("\nUSAGE: largevol txt_file database [userid passwd]\n\n");
  exit( 1 ) ;
     } /* endif */

    /* Abrir el archivo de entrada con los permisos de acceso especificados */

   if ( ( iFileHandle = open(argv[1], iOpenOptions, 0666 ) ) == -1 ) {
     printf( "Error: Could not open %s.\n", argv[2] ) ;
     exit( 2 ) ;
  }

  /* Establecer escapes por error y fin de tabla */

  EXEC SQL WHENEVER SQLERROR GO TO ext ;
  EXEC SQL WHENEVER NOT FOUND GO TO cls ;

  /* Declarar y abrir el cursor */

  EXEC SQL DECLARE c1 CURSOR FOR
           SELECT firstnme, lastname, job FROM employee
           WHERE workdept IS NOT NULL
           AND NODENUMBER(lastname) = CURRENT NODE;
  EXEC SQL OPEN c1 ;

  /* Establecer almacenamiento intermedio temporal para almacenar el
                           resultado captado */

  if ( ( file_buf = ( char * ) malloc( BUF_SIZE ) ) == NULL ) {
     printf( "Error: Allocation of buffer failed.\n" ) ;
     exit( 3 ) ;
  }
  memset( file_buf, 0, BUF_SIZE ) ; /* restablecer almacenamiento intermedio */
  buffer_len = 0 ;  /* restablecer longitud del almacenamiento intermedio */
  write_ptr = file_buf ;  /* restablecer el puntero de grabación */
  /* Para cada registro captado realizar lo siguiente */
  /*  - insertarlo en el almac. interm. tras el       */
  /*    registro almacenado previamente               */
  /*  - comprobar si aún queda espacio suficiente en  */
  /*    el almac. int. para el siguiente registro y bloquear/grabar/ */
  /*    desbloquear el archivo e inicializar el almac.*/
  /*    intermedio si no es así                       */

  do {
     EXEC SQL FETCH c1 INTO :first_name, :last_name, :job_code;
      buffer_len += sprintf( write_ptr, "%s %s %s\n",
                             first_name, last_name, job_code );
      buffer_len = strlen( file_buf ) ;
     /* Grabar el contenido del almac. int. en archivo */
     /* si el almac. int. alcanza el límite            */
     if ( buffer_len >= ( BUF_SIZE - MAX_RECORD_SIZE ) ) {
      /*  obtener bloqueo exclusivo de grabación */
      lock_rc = fcntl( iFileHandle, lock_command, &lock );
 if ( lock_rc != 0 ) goto file_lock_err;
 /*  posicionar al final del archivo */
 lock_rc = lseek( iFileHandle, 0, SEEK_END );
```

```
  if ( lock_rc < 0 ) goto file_seek_err;
  /* grabar el almacenamiento intermedio */
  lock_rc = write( iFileHandle,
                              ( void * ) file_buf, buffer_len );
  if ( lock_rc < 0 ) goto file_write_err;
    /* liberar el bloqueo */
   lock_rc = fcntl( iFileHandle, lock_command, &unlock );
   if ( lock_rc != 0 ) goto file_unlock_err;
   file_buf[0] = '\0' ;  /* restablecer almacenamiento intermedio */
    buffer_len = 0 ;  /* restablecer longitud del almacenamiento intermedio */
    write_ptr = file_buf ;  /* restablecer puntero de grabación */
       }
                  else
          write_ptr = file_buf + buffer_len ;  /* siguiente posición de grabación */
       }
    } while (1) ;

cls:
    /* Grabar la última porción de datos en el archivo */
    if (buffer_len > 0) {
       lock_rc = fcntl(iFileHandle, lock_command, &lock);
       if (lock_rc != 0) goto file_lock_err;
      lock_rc = lseek(iFileHandle, 0, SEEK_END);
       if (lock_rc < 0) goto file_seek_err;
       lock_rc = write(iFileHandle, (void *)file_buf, buffer_len);
       if (lock_rc < 0) goto file_write_err;
       lock_rc = fcntl(iFileHandle, lock_command, &unlock);
      if (lock_rc != 0) goto file_unlock_err;
    }
    free(file_buf);
 close(iFileHandle);
    EXEC SQL CLOSE c1;
    exit (0);
 ext:
    if ( SQLCODE != 0 )
       printf( "Error:  SQLCODE = %ld.\n", SQLCODE );
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    EXEC SQL CONNECT RESET;
    if ( SQLCODE != 0 ) {
       printf( "CONNECT RESET Error:  SQLCODE = %ld\n", SQLCODE );
       exit(4);
    }
    exit (5);
 file_lock_err:
    printf("Error: file lock error = %ld.\n",lock_rc);
      /* desbloqueo incondicional del archivo */
   fcntl(iFileHandle, lock_command, &unlock);
    exit(6);
file_seek_err:
    printf("Error: file seek error = %ld.\n",lock_rc);
      /* desbloqueo incondicional del archivo */
   fcntl(iFileHandle, lock_command, &unlock);
   exit(7);
file_write_err:
    printf("Error: file write error = %ld.\n",lock_rc);
      /* desbloqueo incondicional del archivo */
   fcntl(iFileHandle, lock_command, &unlock);
    exit(8);
file_unlock_err:
    printf("Error: file unlock error = %ld.\n",lock_rc);
      /* desbloqueo incondicional del archivo */
   fcntl(iFileHandle, lock_command, &unlock);
    exit(9);
}
```

Este método no sólo es aplicable a la selección de una única tabla, sino que también sirve para consultas más complejas. No obstante, si la consulte requiere

operaciones no distribuidas (es decir, Explain muestra más de una subsección junto a la subsección Coordinator), puede tener como consecuencia que se produzca un número excesivo de procesos en algunas particiones, en caso de que la consulta se ejecute en paralelo en todas las particiones. En esta situación, puede almacenar el resultado de la consulta en una tabla temporal TEMP, en tantas particiones como sea necesario, y luego realizar la extracción final en paralelo desde TEMP.

Si desea extraer todos los empleados, pero únicamente para clasificaciones de trabajo seleccionadas, puede definir la tabla TEMP con los nombres de columna FIRSTNME, LASTNAME y JOB, tal como sigue:

```
   INSERT INTO TEMP
SELECT FIRSTNME, LASTNAME, JOB FROM EMPLOYEE WHERE WORKDEPT IS NOT NULL
       AND EMPNO NOT IN (SELECT EMPNO FROM EMP_ACT WHERE
                         EMPNO<200)
```

A continuación, realizará la extracción paralela sobre TEMP.

Cuando defina la tabla TEMP, tenga en cuenta lo siguiente:

- Si la consulta especifica una agregación GROUP BY, debe definir la clave de particionamiento de TEMP como subconjunto de las columnas GROUP BY.
- La clave de particionamiento de la tabla TEMP debe tener la cardinalidad (es decir, el número de valores diferenciados en el conjunto de respuestas) suficiente para asegurarse de que la tabla se distribuye de forma igualitaria en las particiones en que está definida.
- Cree la tabla TEMP con el atributo NOT LOGGED INITIALLY, luego confirme (COMMIT) la unidad de trabajo que ha creado la tabla para liberar los posibles bloqueos de catálogo adquiridos.
- Cuando utilice la tabla TEMP, debe emitir las sentencias siguientes en una sola unidad de trabajo:
  1. ALTER TABLE TEMP ACTIVATE NOT LOGGED INITIALLY WITH EMPTY TABLE (para vaciar la tabla TEMP y desactivar el registro)
  2. INSERT INTO TEMP SELECT FIRSTNAME...
  3. COMMIT

  Esta técnica le permite insertar un gran conjunto de respuestas en una tabla sin realizar un registro y sin que se produzca ninguna contención de catálogos. Sin embargo, si una tabla tiene activado el atributo NOT LOGGED INITIALLY, se produce una actividad no registrada y una de las siguientes situaciones:
  - Una sentencia falla, causando una retrotracción
  - Se ejecuta una operación ROLLBACK TO SAVEPOINT

  se retrotrae la unidad de trabajo entera (SQL1476N), con lo que la tabla TEMP queda inutilizable. Si así sucede, deberá eliminar y volver a crear la tabla TEMP. Por este motivo, *no* debe utilizar esta técnica para añadir datos a una tabla que no pueda volver a crear fácilmente.

Si necesita que el conjunto final de respuestas (que está formado por el conjunto de respuestas parciales fusionadas de todas las particiones) esté clasificado, puede:

- Especificar la cláusula SORT BY en la sentencia SELECT final
- Realizar una extracción en un archivo separado en cada partición
- Fusionar los archivos separados en un conjunto de salida utilizando, por ejemplo, el mandato **sort -m** de AIX.

Creación de un entorno simulado de bases de datos particionadas

Puede crear un entorno de prueba para sus aplicaciones de entorno particionadas sin configurar un entorno de bases de datos particionadas.

Procedimiento:

Para crear un entorno simulado de bases de datos particionadas:

1. Cree un modelo de diseño de la base de datos con DB2 Enterprise Server Edition.
2. Cree tablas de ejemplo con la cláusula PARTITIONING KEY, que utilizará para distribuir los datos entre las particiones del entorno de producción.
3. Cree y ejecute sus aplicaciones para la base de datos de prueba.

DB2 Enterprise Server Edition impone las restricciones de clave de particionamiento que se aplican en un entorno de bases de datos particionadas y proporciona un útil entorno de prueba para las aplicaciones.

Resolución de problemas

Las secciones siguientes describen cómo solucionar problemas de aplicaciones en un entorno de bases de datos particionadas.

Consideraciones sobre el manejo de errores en entornos de bases de datos particionadas

En un entorno de bases de datos particionadas, DB2® divide las sentencias de SQL en subsecciones, cada una de las cuales se procesa en la partición que contiene los datos relevantes. Como resultado, se puede producir un error en una partición que no tenga acceso a la aplicación. Esta condición no se produce en un entorno de bases de datos no particionadas.

Debe tener en cuenta lo siguiente:
- Errores no graves de no CURSOR (EXECUTE)
- Errores no graves de CURSOR
- Errores graves
- Varias estructuras SQLCA fusionadas
- Cómo identificar la partición que ha devuelto el error

Si una aplicación finaliza de forma anormal debido a un error grave, es posible que se dejen transacciones dudosas en la base de datos. (Una transacción dudosa pertenece a transacciones globales cuando una fase finaliza satisfactoriamente, pero el sistema falla antes de poder completar la siguiente fase, dejando la base de datos en un estado incoherente.)

Conceptos relacionados:

- "Errores graves en entornos de bases de datos particionadas" en la página 711
- "Varias estructuras SQLCA fusionadas" en la página 711
- "Partición que devuelve el error" en la página 712

Tareas relacionadas:

- "Manually resolving indoubt transactions" en la publicación *Administration Guide: Planning*

Errores graves en entornos de bases de datos particionadas

Si se produce un error grave en un entorno de bases de datos particionadas, se producirá una de las siguientes situaciones:

- El gestor de bases de datos de la partición en la que se produce el error se cierra.

  Las unidades de trabajo activas no se retrotraen.

  En esta situación, debe recuperar la partición y las bases de datos que estaban activas en la partición cuando se produjo el cierre.
- Se impone la salida de todos los agentes de la base de datos en la partición en la que se ha producido el error.

  Todas las unidades de trabajo de dicha base de datos se retrotraen.

  En esta situación, la partición de base de datos en la que se ha producido el error se marca como incoherente. Cualquier intento de acceder a la misma da como resultado la devolución de SQLCODE -1034 (SQLSTATE 58031) o SQLCODE -1015 (SQLSTATE 55025). Antes de que el usuario o cualquier otra aplicación de otra partición pueda acceder a esta partición de base de datos, debe ejecutar el mandato RESTART DATABASE contra la base de datos.

El error grave SQLCODE -30081 (SQLSTATE 08001) se puede producir por varios motivos. Si recibe este mensaje, compruebe la SQLCA, la cual indicará qué partición ha fallado. Luego compruebe el archivo de registro de notificaciones del administrador para ver detalles.

Conceptos relacionados:

- "Partición que devuelve el error" en la página 712

Información relacionada:

- "Mandato RESTART DATABASE" en la publicación *Consulta de mandatos*

Varias estructuras SQLCA fusionadas

Distintos agentes de distintas partición de base de datos pueden ejecutar una sentencia de SQL, y cada agente puede devolver una SQLCA distinta para diferentes errores o avisos. El agente coordinador también tiene su propia SQLCA. Además, la SQLCA también tiene campos que indican números globales (como los campos *sqlerrd* que indican números de filas). Para ofrecer una vista coherente para las aplicaciones, todos los valores de SQLCA se fusionan en una estructura.

La notificación de errores se realiza del siguiente modo:

- Las condiciones de errores graves siempre se notifican. En cuanto se notifica un error grave, no se realizan adiciones, además del error grave, a la SQLCA.
- Si no se produce ningún error grave, un error de punto muerto tiene precedencia sobre los demás errores.
- Para los demás errores, se devuelve a la aplicación la SQLCA correspondiente al primer SQLCODE negativo.
- Si no se detecta ningún valor SQLCODE negativo, se devuelve a la aplicación la SQLCA correspondiente al primer aviso (es decir, un SQLCODE positivo). Se produce una excepción a esta norma si se emite una operación de manipulación de datos en una tabla que está vacía en una partición pero que tiene datos en otras particiones. El SQLCODE +100 sólo se devuelve a la aplicación si los

agentes de todas las particiones devuelven SQL0100W porque la tabla está vacía en todas las particiones o porque no hay filas que satisfagan la cláusula WHERE de una sentencia UPDATE.
- Para todos los errores y avisos, el campo *sqlwarn* contiene los distintivos de aviso recibidos de todos los agentes.
- Los valores de los campos *sqlerrd* que indican números de filas son sumas de todos los agentes.

Una aplicación puede recibir un error o aviso siguiente después de que se corrija el problema que ocasionó el primer error o aviso. Los errores se notifican a la SQLCA para asegurar que el primer error detectado tiene prioridad sobre otros. Esto asegura que un error ocasionado por un error anterior no puede sobregrabar el error original. Los errores graves y los errores de punto muerto tienen una prioridad más alta porque necesitan una acción inmediata por parte del agente coordinador.

Información relacionada:
- "SQLCA" en la publicación *Administrative API Reference*

Partición que devuelve el error

Si una partición devuelve un error o aviso, su número está en el campo SQLERRD(6) de la SQLCA. El número de este campo coincide con el especificado para la partición en el archivo `db2nodes.cfg`.

Si una llamada a una sentencia de SQL o API resulta satisfactoria, el número de partición de este campo no es significativo.

Información relacionada:
- "SQLCA" en la publicación *Administrative API Reference*

Aplicaciones en bucle o suspendidas

Es posible que, después de iniciar una consulta o aplicación, sospeche que está suspendida (no muestra ninguna actividad) o que ha entrado en un bucle (muestra actividad, pero no se devuelve ningún resultado a la aplicación). Asegúrese de que ha activado los tiempos de espera de bloqueo. Sin embargo, en algunas situaciones no se devuelve ningún error. En estas situaciones, pueden resultarle de ayuda las herramientas de diagnóstico que se proporcionan con DB2® y la instantánea del supervisor del sistema de bases de datos.

Una de las funciones del supervisor del sistema de bases de datos que resulta útil para depurar aplicaciones consiste en visualizar el estado de todos los agentes activos. Para sacar el máximo provecho de una instantánea, asegúrese de que la recopilación de sentencias se realiza antes de ejecutar la aplicación (preferiblemente inmediatamente después de ejecutar DB2START) del siguiente modo:

```
db2_all "db2 UPDATE MONITOR SWITCHES USING STATEMENT ON"
```

Si sospecha que la aplicación o consulta está atascada o ha entrado en un bucle, emita el siguiente mandato:

```
db2_all "db2 GET SNAPSHOT FOR AGENTS ON base de datos
```

Conceptos relacionados:
- "Database system monitor" en la publicación *System Monitor Guide and Reference*

- "The database system monitor information" en la publicación *Administration Guide: Performance*

Información relacionada:

- "Mandato GET SNAPSHOT" en la publicación *Consulta de mandatos*
- "Mandato UPDATE MONITOR SWITCHES" en la publicación *Consulta de mandatos*
- "db2trc - Mandato Rastrear" en la publicación *Consulta de mandatos*
- "db2support - Mandato Herramienta de análisis de problemas y recolección del entorno" en la publicación *Consulta de mandatos*

Capítulo 32. Técnicas comunes de aplicación de DB2

Ejecución de aplicaciones desde Local System Account de Windows. 715
Columnas generadas 715
Columnas de identidad 716
Recuperación de conjuntos de resultados a partir de una sentencia de cambio de datos de SQL. . . 717
Tablas de resultados intermedios 718
Vistas y tablas de destino 719
Ordenación del conjunto de resultados basada en INPUT SEQUENCE 719
Recuperación de conjuntos de resultados a partir de sentencias de cambio de datos de SQL utilizando cursores 720
Incluir columnas 721
Incluir columnas en operaciones INSERT . . . 721
Incluir columnas en operaciones UPDATE and DELETE 722
Operaciones UPDATE, INSERT, DELETE y MERGE de búsqueda frente a selecciones completas . . . 722
Valores secuenciales y objetos de secuencia . . . 723
Generación de valores secuenciales 723
Gestión del comportamiento de secuencias . . 725
Rendimiento de la aplicación y objetos de secuencia 726
Comparación entre objetos de secuencia y columnas de identidad 726
Tablas temporales declaradas y rendimiento de la aplicación 726
Transmisión de grandes volúmenes de datos a través de una red 729

Ejecución de aplicaciones desde Local System Account de Windows

En las plataformas Windows, DB2 soporta la ejecución de aplicaciones desde Local System Account (LSA) utilizando una conexión implícita de base de datos local. No puede utilizar LSA para establecer una conexión explícita de base de datos ni una conexión remota de base de datos.

En DB2, el nombre de esquema para LSA es SYSTEM. Debido a que DB2 tiene restricciones respecto al uso de objetos cuyo nombre de esquema comience con "SYS", las aplicaciones de base de datos que se ejecutan desde LSA no pueden crear objetos DB2 utilizando el esquema de conexión implícita. Siga estos pasos para crear objetos DB2 de una aplicación que se ejecutará desde LSA:

- Para el SQL estático, vincule la aplicación con un valor para QUALIFIER que no sea "SYSTEM".
- Para el SQL dinámico, debe utilizar un nombre de esquema diferente de "SYSTEM" para calificar explícitamente los objetos que está creando, o asignar al registro CURRENT SCHEMA un nombre de esquema que no sea "SYSTEM".

Información relacionada:

- "Sentencia SET SCHEMA" en la publicación *Consulta de SQL, Volumen 2*
- "Mandato BIND" en la publicación *Consulta de mandatos*
- "Registro especial CURRENT SCHEMA" en la publicación *Consulta de SQL, Volumen 1*

Columnas generadas

En lugar de utilizar engorrosos activadores de inserción y actualización, DB2® le permite incluir columnas generadas en sus tablas mediante la cláusula GENERATED ALWAYS AS. Una columna generada es una columna que deriva de los valores de cada fila procedente de una expresión, en lugar de hacerlo de una operación de inserción o actualización. Aunque que la combinación de un activador de actualización y un activador de inserción puede conseguir un efecto parecido, la utilización de una columna generada garantiza que el valor derivado es coherente con la expresión.

Para crear una columna generada en una tabla, utilice la cláusula GENERATED ALWAYS AS para la columna e incluya la expresión de la cual se derivará el valor correspondiente a la columna. Puede incluir la cláusula GENERATED ALWAYS AS en sentencias ALTER TABLE y CREATE TABLE. El siguiente ejemplo crea una tabla con dos columnas normales, "c1" y "c2", y dos columnas generadas, "c3" y "c4", que se derivan de las columnas normales de la tabla.

```
CREATE TABLE T1(c1 INT, c2 DOUBLE,
                  c3 DOUBLE GENERATED ALWAYS AS (c1 + c2),
                  c4 SMALLINT GENERATED ALWAYS AS
                    (CASE
                       WHEN c1 > c2 THEN 1
                       ELSE NULL
                     END)
                );
```

Tareas relacionadas:

- "Defining a generated column on a new table" en la publicación *Administration Guide: Implementation*
- "Defining a generated column on an existing table" en la publicación *Administration Guide: Implementation*

Información relacionada:

- "Sentencia ALTER TABLE" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE TABLE" en la publicación *Consulta de SQL, Volumen 2*

Ejemplos relacionados:

- "TbGenCol.java -- How to use generated columns (JDBC)"

Columnas de identidad

Las columnas de identidad proporcionan a los programadores de aplicaciones de DB2® una forma sencilla de generar automáticamente un valor numérico de columna para cada fila de una tabla. Puede hacer que este valor se genere como valor exclusivo y luego definir la columna de identidad como la clave principal para la tabla. Para crear una columna de identidad, incluya la cláusula IDENTITY en la sentencia CREATE TABLE.

Utilice columnas de identidad en las aplicaciones para evitar problemas de concurrencia y de rendimiento que se pueden producir cuando una aplicación genera su propio contador exclusivo fuera de la base de datos. Cuando no utilice columnas de identidad para generar automáticamente claves principales exclusivas, un diseño común consiste en almacenar un contador en una tabla con una sola fila. Luego cada transacción bloquea esta tabla, aumenta el número y luego confirma la transacción para desbloquear el contador. Desgraciadamente, este diseño sólo permite que una sola transacción aumente el contador cada vez.

Por el contrario, si utiliza una columna de identidad para generar automáticamente claves principales, la aplicación puede conseguir niveles de concurrencia muy superiores. Con columnas de identidad, DB2 mantiene el contador de forma que las transacciones no lo tienen que bloquear. Las aplicaciones que utilizan columnas de identidad pueden funcionar mejor porque una transacción no confirmada que ha aumentado el contador no evita que otras transacciones siguientes puedan también aumentar el contador.

El contador correspondiente a la columna de identidad aumenta o desciende independientemente de la transacción. Si una determinada transacción aumenta un contador de identidad dos veces, dicha transacción puede observar un salto en los dos números que se generan porque es posible que otras transacciones estén incrementando simultáneamente el mismo contador de identidad.

Puede que parezca que una columna de identidad ha generado saltos en el contador, como resultado de una transacción que se ha retrotraído o porque la base de datos ha colocado en antememoria un rango de valores que se han desactivado (de forma normal o anómala) antes de que se asignaran todos los valores colocados en antememoria.

Para recuperar el valor generado después de insertar una fila nueva en una tabla con una columna de identidad, utilice la función `identity_val_local()`.

La cláusula IDENTITY está disponible para las sentencias CREATE TABLE y ALTER TABLE.

Conceptos relacionados:

- "Identity columns" en la publicación *Administration Guide: Planning*

Tareas relacionadas:

- "Defining an identity column on a new table" en la publicación *Administration Guide: Implementation*
- "Modifying an identity column definition" en la publicación *Administration Guide: Implementation*
- "Altering an identity column" en la publicación *Administration Guide: Implementation*

Información relacionada:

- "Sentencia ALTER TABLE" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE TABLE" en la publicación *Consulta de SQL, Volumen 2*

Ejemplos relacionados:

- "tbident.sqc -- How to use identity columns (C)"
- "TbIdent.java -- How to use Identity Columns (JDBC)"
- "TbIdent.sqlj -- How to use Identity Columns (SQLj)"

Recuperación de conjuntos de resultados a partir de una sentencia de cambio de datos de SQL

Las aplicaciones que modifican tablas con sentencias INSERT, UPDATE o DELETE pueden requerir el proceso adicional de las filas modificadas. Para facilitar este proceso, puede intercalar operaciones de cambio de datos de SQL en la cláusula FROM de las sentencias SELECT y SELECT INTO. Dentro de una única unidad de trabajo, las aplicaciones pueden recuperar un conjunto de resultados que contenga las filas modificadas a partir de una vista o tabla modificada mediante una operación de cambio de datos de SQL.

Por ejemplo, la sentencia siguiente actualiza los salarios de todos los registros de la tabla EMPLOYEE de la base de datos SAMPLE y después devuelve el número de empleado y el salario nuevo para todas las filas actualizadas.

```
SELECT empno, salary FROM FINAL TABLE
 (UPDATE employee SET salary = salary * 1.10 WHERE job = 'CLERK')
```

Para devolver datos satisfactoriamente, las sentencias SELECT que recuperan conjuntos de datos a partir de (FROM) operaciones de cambio de datos de SQL requieren que las operaciones de cambio de datos de SQL se ejecuten satisfactoriamente. El final satisfactorio de las operaciones de cambio de datos de SQL incluye el proceso de todas las restricciones y activadores, si es de aplicación.

Por ejemplo, suponga que un usuario con privilegios SELECT, pero sin privilegios INSERT en la tabla EMPLOYEE intenta efectuar una sentencia SELECT FROM INSERT en la tabla EMPLOYEE. La operación INSERT falla debido a los privilegios que faltan y, como consecuencia, falla toda la sentencia SELECT.

Tenga en cuenta la consulta siguiente, en la que se seleccionan los registros de la tabla EMPLOYEE y después se insertan en una tabla diferente, denominada EMP. Esta sentencia SELECT fallará.

```
SELECT empno FROM FINAL TABLE
 (INSERT INTO emp(name, salary)
 SELECT firstnme || midinit || lastname, salary
 FROM employee)
```

Si la tabla EMPLOYEE tiene 100 filas y la fila 90 tiene un valor de SALARY de $9.999.000,00, la suma de $10.000,00 haría que se produjera un desbordamiento decimal. El desbordamiento hará que el gestor de la base de datos retrotraiga las inserciones a la tabla EMP.

Tablas de resultados intermedios

Las filas modificadas de la vista o tabla que es el destino de una operación de cambio de datos de SQL en la cláusula FROM de una sentencia SELECT componen una tabla de resultados intermedios. La tabla de resultados intermedios incluye todas las columnas de la vista o tabla de destino, además de las acciones de incluir columnas definidas en la operación de cambio de datos de SQL. Puede hacer referencia a todas las columnas de una tabla de resultados intermedios por su nombre en la lista de selección, por medio de la cláusula ORDER BY o por medio de la cláusula WHERE.

El contenido de la tabla de resultados intermedios depende del calificador especificado en la cláusula FROM. Deberá incluir uno de los siguientes calificadores de cláusula FROM en las sentencias SELECT que recuperan conjuntos de resultados como tablas de resultados intermedios.

OLD TABLE
Las filas de la tabla de resultados intermedios contendrán valores de las filas de tablas de destino en el punto que precede de modo inmediato a la ejecución de activadores anteriores y la operación de cambio de datos de SQL. El calificador OLD TABLE se aplica a las operaciones UPDATE y DELETE.

NEW TABLE
Las filas de la tabla de resultados intermedios contendrán valores de las filas de tablas de destino en el punto que sigue de modo inmediato a la ejecución de la sentencia de cambio de datos de SQL, pero antes de la evaluación de la integridad referencial y de que se active cualquier activador posterior. El calificador NEW TABLE se aplica a las operaciones UPDATE y INSERT.

FINAL TABLE
Este calificador devuelve la misma tabla de resultados intermedios que NEW TABLE. Además, la utilización de FINAL TABLE garantiza que ningún activador posterior ni restricción de integridad referencial modificará aún más el destino de la operación de UPDATE o INSERT. El calificador FINAL TABLE se aplica a las operaciones UPDATE e INSERT.

Los calificadores de la cláusula FROM determinan que la versión de los datos de destino está en la tabla de resultados intermedios. Estos calificadores no afectan a la inserción, supresión o actualizaciones de las filas de la tabla de destino.

Vistas y tablas de destino

Al seleccionar conjuntos de resultados a partir de (FROM) operaciones de cambio de datos de SQL, el destino puede ser una tabla o una vista.

En las operaciones de cambio de datos de SQL, la tabla de resultados no puede incluir filas que ya no satisfagan la definición de vista para NEW TABLE y FINAL TABLE. Si intercala una sentencia INSERT o UPDATE que haga referencia a una vista de una sentencia SELECT, la vista debe definirse como WITH CASCADED CHECK OPTION. De modo alternativo, la vista debe satisfacer las restricciones que le permitirían definirlo como WITH CASCADED CHECK OPTION.

Si el destino de las operaciones de cambio de datos de SQL intercaladas en la cláusula FROM de una sentencia SELECT es fullselect, la tabla de resultados puede incluir filas que ya no se califiquen en fullselect. Esto se debe a que los predicados de la cláusula WHERE no vuelven a evaluarse frente a los valores actualizados.

Ordenación del conjunto de resultados basada en INPUT SEQUENCE

Para seleccionar (SELECT) filas en el mismo orden en el que se insertaron en la vista o tabla de destino, utilice las palabras clave INPUT SEQUENCE de la cláusula ORDER BY. La utilización de las palabras clave INPUT SEQUENCE no hace que las filas se inserten en el mismo orden en que se facilitaron.

El ejemplo siguiente demuestra la utilización de las palabras clave INPUT SEQUENCE en la cláusula ORDER BY para ordenar el conjunto de resultados de una operación INSERT.

```
CREATE TABLE orders (purchase_date DATE,
                     sales_person VARCHAR(16),
                     region VARCHAR(10),
                     quantity SMALLINT,
                     order_num INTEGER NOT NULL
                       GENERATED ALWAYS AS IDENTITY (START WITH 100,
                                     INCREMENT BY 1))

SELECT * FROM FINAL TABLE
         (INSERT INTO orders
          (purchase_date, sales_person, region, quantity)
          VALUES (CURRENT DATE,'Judith','Beijing',6),
                 (CURRENT DATE,'Marieke','Medway',5),
                 (CURRENT DATE,'Hanneke','Halifax',5))
         ORDER BY INPUT SEQUENCE

PURCHASE_DATE SALES_PERSON     REGION     QUANTITY ORDER_NUM
------------- ---------------- ---------- -------- -----------
07/18/2003    Judith           Beijing           6         100
07/18/2003    Marieke          Medway            5         101
07/18/2003    Hanneke          Halifax           5         102
```

También puede ordenar las filas del conjunto de resultados utilizando las acciones de incluir columnas.

Conceptos relacionados:

- "Recuperación de conjuntos de resultados a partir de sentencias de cambio de datos de SQL utilizando cursores" en la página 720
- "Incluir columnas" en la página 721

Información relacionada:

- "Sentencia DELETE" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia INSERT" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia SELECT" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia SELECT INTO" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia UPDATE" en la publicación *Consulta de SQL, Volumen 2*

Recuperación de conjuntos de resultados a partir de sentencias de cambio de datos de SQL utilizando cursores

Puede declarar cursores para las consultas que recuperan conjuntos de resultados a partir de las operaciones de cambio de datos de SQL. Por ejemplo:

```
DECLARE C1 CURSOR FOR SELECT salary FROM FINAL TABLE
        (INSERT INTO employee (name, salary, level)
            SELECT name, income, band FROM old_employee)
```

Los errores que se produzcan al efectuar la extracción desde un cursor cuya definición contenga una operación de cambio de datos de SQL no ocasionarán una retrotracción de las filas modificadas. Aún en el caso de que los errores den como resultado que se cierre el cursor, las modificaciones de la fila permanecerán intactas debido a que éstas se completaron cuando la aplicación abrió el cursor.

Al abrir tal cursor, el gestor de la base de datos ejecuta completamente la operación de cambio de datos de SQL y el conjunto de resultados se almacena en una tabla temporal. Si se produce un error mientras se abre el cursor, se retrotraerán los cambios efectuados por la operación de cambio de datos de SQL. Las actualizaciones ulteriores en la vista o tabla de destino no aparecerán en las filas de tabla de resultados para los cursores que recuperan conjuntos de resultados a partir de las operaciones de cambio de datos de SQL. Por ejemplo, una aplicación declara un cursor, abre el cursor, efectúa una extracción, actualiza la tabla y extrae filas adicionales. Las extracciones posteriores a la sentencia UPDATE devolverán los valores determinados durante el proceso de apertura del cursor antes de la sentencia UPDATE.

Puede declarar cursores desplazables para las consultas que recuperen conjuntos de resultados a partir de las operaciones de cambio de datos de SQL. Puesto que la tabla de resultados se genera cuando se abre (OPEN) el cursor, las modificaciones de datos ya se han grabado en la vista o tabla de destino. Los cursores con consultas que seleccionan filas a partir de una operación de cambio de datos de SQL deben definirse como INSENSITIVE o ASENSITIVE.

Nota: Los cursores desplegables sólo se soportan en las aplicaciones CLI, JDBC y SQLJ.

Si declara un cursor con la opción WITH HOLD y la aplicación realiza una operación de confirmación (COMMIT), se confirmarán todos los cambios de datos.

Los cursores que no se declaran como WITH HOLD se comportan de la misma manera. Para todos los cursores, la operación de cambio de datos de SQL incluida en la consulta se evaluará completamente antes de que se extraiga cualquier fila.

Al realizar una retrotracción explícita para una sentencia OPEN CURSOR o al efectuar una retrotracción a un punto para guardar anterior a una sentencia OPEN CURSOR, se deshará la totalidad de los cambios de datos para dicho cursor. Para los cursores con consultas que recuperan conjuntos de resultados a partir de operaciones de cambios de datos de SQL, todos los cambios de datos se desharán después de una retrotracción, pero se retendrá el cursor y podrán seguir extrayéndose las filas insertadas previamente.

Conceptos relacionados:

- "Recuperación de conjuntos de resultados a partir de una sentencia de cambio de datos de SQL" en la página 717
- "Incluir columnas" en la página 721

Información relacionada:

- "Sentencia DECLARE CURSOR" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia DELETE" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia FETCH" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia OPEN" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia SELECT" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia UPDATE" en la publicación *Consulta de SQL, Volumen 2*

Incluir columnas

Con la acción de incluir columnas, puede introducir columnas que no existan en la vista o tabla de destino de una tabla de resultados intermedios. Puede asignar valores para incluir las columnas a utilizar como manejadores para las filas de la tabla de resultados intermedios. La acción de incluir columnas no afecta a las operaciones de cambio de datos de SQL, ni cambia las definiciones de vistas o tablas de destino.

Una acción de incluir columna puede ser de cualquier tipo de datos, puede anularse y debe tener un nombre que resulte exclusivo respecto a cualquier columna de la vista o tabla de destino de la operación de cambio de datos de SQL. Puede consultar la acción de incluir columnas en la lista de selección, la cláusula ORDER BY o la cláusula WHERE de la sentencia SELECT. En los conjuntos de resultados, las acciones de incluir columnas aparecen como las columnas que hay más a la derecha.

Incluir columnas en operaciones INSERT

Para asignar valores a la acción de incluir columnas en operaciones INSERT, podrá utilizar la cláusula VALUES. Una utilización habitual de la acción de incluir columnas en las operaciones INSERT es la de personalizar el orden de los conjuntos de resultados. Por ejemplo:

```
SELECT * FROM FINAL TABLE
         (INSERT INTO sales INCLUDE (sortkey integer) VALUES
          (CURRENT DATE,'Judith', 'Halifax',6,1),
          (CURRENT DATE,'Marieke', 'Medway',5,3),
          (CURRENT DATE,'Hanneke', 'Halifax',5,2))
         ORDER BY sortkey
```

```
SALES_DATE SALES_PERSON    REGION          SALES       SORTKEY
---------- --------------- --------------- ----------- -----------
07/16/2003 Judith          Amsterdam                 6           1
07/16/2003 Hanneke         Halifax                   5           2
07/16/2003 Marieke         Medway                    5           3
```

También puede asignar valores a una acción de incluir columna en una operación INSERT utilizando una acción de fullselect.

Incluir columnas en operaciones UPDATE and DELETE

Para asignar valores a la acción de incluir columnas en operaciones UPDATE o DELETE, utilice la cláusula SET. Si no se asigna ningún valor a una acción de incluir columna en la cláusula SET de una sentencia UPDATE o DELETE, se devolverá un valor NULL para dicha columna.

En las sentencias UPDATE, podrá utilizar la acción de incluir columnas para devolver los valores de columna nuevos y antiguos para una fila. Por ejemplo:

```
SELECT salary, oldSalary FROM FINAL TABLE
       (UPDATE employee INCLUDE (oldSalary decimal(9,2))
        SET oldSalary = salary, salary = salary * 1.05
        WHERE job = 'CLERK')

SALARY      OLDSALARY
----------- -----------
   30712.50    29250.00
   23289.00    22180.00
   30198.00    28760.00
   20139.00    19180.00
   18112.50    17250.00
   28749.00    27380.00
```

Conceptos relacionados:

- "Recuperación de conjuntos de resultados a partir de una sentencia de cambio de datos de SQL" en la página 717
- "Recuperación de conjuntos de resultados a partir de sentencias de cambio de datos de SQL utilizando cursores" en la página 720

Información relacionada:

- "Sentencia DELETE" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia INSERT" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia SELECT" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia UPDATE" en la publicación *Consulta de SQL, Volumen 2*

Operaciones UPDATE, INSERT, DELETE y MERGE de búsqueda frente a selecciones completas

A partir de DB2® Versión 8.1.4 puede emitir sentencias de búsqueda INSERT, UPDATE, DELETE y MERGE sobre los resultados de operaciones de selección completa. Esta función le permite convertir en una sola sentencia una tarea que en otro caso necesitaría dos sentencias para ejecutarse (una selección completa y una sentencia INSERT, UPDATE, DELETE o MERGE sobre los resultados de la selección completa). La combinación de tareas en una sola sentencia reduce la posibilidad de puntos muertos y puede hacer innecesario el escribir definiciones de vista y de cursor. Cualquier consulta que se pueda utilizar para producir una vista susceptible de insertar, actualizar o suprimir puede ser objeto de una sentencia INSERT, UPDATE, DELETE o MERGE.

Por ejemplo, la sentencia siguiente suprime los diez empleados de la tabla EMPLOYEE que tengan el nivel de formación (edlevel) más bajo.

```
DELETE FROM (SELECT edlevel FROM employee
               ORDER BY edlevel
               FETCH FIRST 10 ROWS ONLY)
```

Información relacionada:

- "Sentencia DELETE" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia INSERT" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia SELECT" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia SELECT INTO" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia UPDATE" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia MERGE" en la publicación *Consulta de SQL, Volumen 2*

Valores secuenciales y objetos de secuencia

Las secciones siguientes describen consideraciones sobre valores secuenciales y objetos de secuencia.

Generación de valores secuenciales

La generación de valores secuenciales es un problema común del desarrollo de aplicaciones de base de datos. La mejor solución a este problema es la utilización de objetos de secuencia y expresiones de secuencia en SQL. Cada *objeto de secuencia* es un objeto de base de datos con nombre exclusivo al que sólo pueden acceder las expresiones de secuencia. Hay dos *expresiones de secuencia*: la expresión PREVVAL y la expresión NEXTVAL. La expresión PREVVAL devuelve el último valor generado en el proceso de aplicación correspondiente al objeto de secuencia especificado. Cualquier expresión NEXTVAL que aparezca en la misma sentencia que la expresión PREVVAL no tiene ningún efecto sobre el valor generado por la expresión PREVAL en esa sentencia. La expresión de secuencia NEXTVAL incrementa el valor del objeto de secuencia y devuelve el nuevo valor del objeto de secuencia.

Para crear un objeto de secuencia, emita la sentencia CREATE SEQUENCE. Por ejemplo, para crear un objeto de secuencia llamado id_values utilizando los atributos por omisión, emita la sentencia siguiente:

```
  CREATE SEQUENCE id_values
```

Para generar el primer valor de la sesión de la aplicación correspondiente al objeto de secuencia, emita una sentencia VALUES que utilice la expresión NEXTVAL:

```
VALUES NEXTVAL FOR id_values

1
-----------
          1

            1 registro(s) seleccionado(s).
```

Para visualizar el valor actual del objeto de secuencia, emita una sentencia VALUES que utilice la expresión PREVVAL:

```
VALUES PREVVAL FOR id_values

1
-----------
```

```
          1

            1 registro(s) seleccionado(s).
```

Se puede recuperar el valor actual del objeto de secuencia repetidamente y el valor que el objeto de secuencia devuelve no cambia hasta que se emite una expresión NEXTVAL. En el ejemplo siguiente, la expresión PREVVAL devuelve un valor de 1, hasta que la expresión NEXTVAL de la conexión actual aumenta el valor del objeto de secuencia:

```
VALUES PREVVAL FOR id_values

1
-----------
          1

            1 registro(s) seleccionado(s).

VALUES PREVVAL FOR id_values

1
-----------
          1

            1 registro(s) seleccionado(s).

VALUES NEXTVAL FOR id_values

1
-----------
          2

            1 registro(s) seleccionado(s).

VALUES PREVVAL FOR id_values

1
-----------
          2

            1 registro(s) seleccionado(s).
```

Para actualizar el valor de una columna con el siguiente valor del objeto de secuencia, incluya la expresión NEXTVAL en la sentencia UPDATE, del modo siguiente:

```
UPDATE personal
  SET id = NEXTVAL FOR id_values
  WHERE id = 350
```

Para insertar una nueva fila en una tabla utilizando el siguiente valor del objeto de secuencia, incluya la expresión NEXTVAL en la sentencia INSERT, del modo siguiente:

```
INSERT INTO personal (id, nombre, dept, cargo)
  VALUES (NEXTVAL FOR id_values, 'Kandil', 51, 'Director')
```

Información relacionada:

- "Sentencia CREATE SEQUENCE" en la publicación *Consulta de SQL, Volumen 2*

Ejemplos relacionados:

- "DbSeq.java -- How to create, alter and drop a sequence in a database (JDBC)"

Gestión del comportamiento de secuencias

Puede adaptar el comportamiento de objetos de secuencia para satisfacer las necesidades de su aplicación. Se cambian los atributos de un objeto de secuencia cuando se emite la sentencia CREATE SEQUENCE para crear un nuevo objeto de secuencia y cuando se emite una sentencia ALTER SEQUENCE para un objeto de secuencia ya existente. A continuación se encuentran algunos de los atributos de un objeto de secuencia que se pueden especificar:

Tipo de datos
La cláusula AS de la sentencia CREATE SEQUENCE especifica el tipo de datos numérico del objeto de secuencia. El tipo de datos determina los posibles valores mínimo y máximo del objeto de secuencia (los valores mínimo y máximo correspondientes a un tipo de datos se listan en el tema que describe los límites de SQL). No se puede cambiar el tipo de datos de un objeto de secuencia; en su lugar, deberá descartar el objeto de secuencia emitiendo la sentencia DROP SEQUENCE y emitir una sentencia CREATE SEQUENCE con el nuevo tipo de datos.

Valor inicial
La cláusula START WITH de la sentencia CREATE SEQUENCE establece el valor inicial del objeto de secuencia. La cláusula RESTART WITH de la sentencia ALTER SEQUENCE restablece el valor del objeto de secuencia en un valor especificado.

Valor mínimo
La cláusula MINVALUE establece el valor mínimo del objeto de secuencia.

Valor máximo
La cláusula MAXVALUE establece el valor máximo del objeto de secuencia.

Valor de incremento
La cláusula INCREMENT BY establece el valor que cada expresión NEXTVAL añade al valor actual del objeto de secuencia. Para disminuir el valor del objeto de secuencia, especifique un valor negativo.

Secuencia cíclica
La cláusula CYCLE hace que el valor de un objeto de secuencia que alcanza su valor máximo o mínimo genere su valor mínimo o máximo respectivo en la siguiente expresión NEXTVAL.

Por ejemplo, para crear un objeto de secuencia llamado id_values que empiece con un valor mínimo de 0, tenga un valor máximo de 1000, incremente en 2 con cada expresión NEXTVAL y vuelva a su valor mínimo cuando se alcance el valor máximo, emita la sentencia siguiente:

```
CREATE SEQUENCE id_values
  START WITH 0
  INCREMENT BY 2
  MAXVALUE 1000
  CYCLE
```

Información relacionada:

- "Límites de SQL" en la publicación *Consulta de SQL, Volumen 1*
- "Sentencia ALTER SEQUENCE" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE SEQUENCE" en la publicación *Consulta de SQL, Volumen 2*

Rendimiento de la aplicación y objetos de secuencia

Al igual que las columnas de identidad, la utilización de objetos de secuencia para generar valores normalmente mejora el rendimiento de las aplicaciones en comparación con los acercamientos alternativos. La alternativa a los objetos de secuencia consiste en crear una tabla de una sola columna que almacene el valor actual e incremente este valor con un activador o bajo el control de la aplicación. En un entorno distribuido donde las aplicaciones acceden de forma simultánea a la tabla de una sola columna, el bloqueo necesario para forzar el acceso en serie a la tabla puede afectar gravemente al rendimiento.

Los objetos de secuencia evitan los problemas de bloqueo que están asociados con el enfoque de tablas de una única columna y pueden almacenar en antememoria valores de secuencia en memoria para mejorar el tiempo de respuesta de DB2®. Para maximizar el rendimiento de aplicaciones que utilizan objetos de secuencia, asegúrese de que el objeto de secuencia almacene en antememoria una cantidad apropiada de valores de secuencia. La cláusula CACHE de las sentencias CREATE SEQUENCE y ALTER SEQUENCE especifica el número máximo de valores de secuencia que DB2 genera y almacena en memoria.

Si el objeto de secuencia debe generar valores en orden, sin incorporar espacios en ese orden debido a una anomalía del sistema o desactivación de la base de datos, utilice las cláusulas ORDER y NO CACHE en la sentencia CREATE SEQUENCE. La cláusula NO CACHE garantiza que no aparezcan espacios en los valores generados a expensas de parte del rendimiento de la aplicación, puesto que fuerza al objeto de secuencia a grabar en la anotación cronológica de la base de datos cada vez que genera un nuevo valor. Observe que es posible que sigan apareciendo saltos debido a transacciones que se retrotraen y no utilizan realmente el valor de secuencia que han solicitado.

Comparación entre objetos de secuencia y columnas de identidad

Aunque los objetos de secuencia y las columnas de identidad parecen tener finalidades parecidas para aplicaciones de DB2®, existe una diferencia importante. Una columna de identidad genera automáticamente valores para una columna en una sola tabla. Un objeto de secuencia genera valores secuenciales bajo petición que se pueden utilizar en cualquier sentencia de SQL.

Tablas temporales declaradas y rendimiento de la aplicación

Una *tabla temporal declarada* es una tabla temporal a la que sólo pueden acceder las sentencias de SQL que emite la aplicación que ha creado la tabla temporal. Una tabla temporal declarada no dura más que la conexión de la aplicación con la base de datos.

Utilice tablas temporales declaradas para mejorar potencialmente el rendimiento de sus aplicaciones. Cuando crea una tabla temporal declarada, DB2® no inserta una entrada en las tablas del catálogo del sistema, y por lo tanto el servidor no experimenta problemas relacionados con la contención por el catálogo. En comparación con las tablas normales, DB2 no bloquea las tablas temporales declaradas ni sus filas y, si especifica el parámetro NOT LOGGED al crearlas, no registra las tablas temporales declaradas ni su contenido. Si la aplicación actual crea tablas para procesar grandes cantidades de datos y elimina dichas tablas

cuando la aplicación ha terminado de manipular dichos datos, considere la posibilidad de utilizar tablas temporales declaradas en lugar de tablas normales.

Si desarrolla aplicaciones escritas para usuarios simultáneos, las aplicaciones se pueden beneficiar de las tablas temporales declaradas. A diferencia de las tablas normales, las tablas temporales declaradas no están sujetas a colisión de nombres. Para cada instancia de la aplicación, DB2 puede crear una tabla temporal declarada con un nombre idéntico. Por ejemplo, para escribir una aplicación para usuarios simultáneos que utilice tablas normales para procesar grandes cantidades de datos temporales, debe asegurarse de que cada instancia de la aplicación utilice un nombre exclusivo para la tabla normal que contiene los datos temporales. Normalmente, crearía otra tabla que efectúe un seguimiento de los nombres de las tablas que se están utilizando en cada momento. Con tablas temporales declaradas, simplemente especifique un nombre de tabla temporal declarada para sus datos temporales. DB2 garantiza que cada instancia de la aplicación utiliza una tabla exclusiva.

Para utilizar una tabla temporal declarada, siga los pasos siguientes:

Paso 1. Asegúrese de que existe un USER TEMPORARY TABLESPACE. Si no existe ningún USER TEMPORARY TABLESPACE, emita una sentencia CREATE USER TEMPORARY TABLESPACE.

Paso 2. Emita una sentencia DECLARE GLOBAL TEMPORARY TABLE en la aplicación.

El esquema correspondiente a tablas temporales declaradas siempre es SESSION. Para utilizar la tabla temporal declarada en las sentencias de SQL, debe hacer referencia a la tabla mediante el calificador de esquema SESSION de forma explícita o utilizando un esquema DEFAULT de SESSION para calificar las referencias no calificadas. En el siguiente ejemplo, el nombre de tabla siempre está calificado mediante el nombre de esquema SESSION cuando crea una tabla temporal declarada denominada TT1 con la siguiente sentencia:

```
DECLARE GLOBAL TEMPORARY TABLE TT1
```

Para seleccionar el contenido de la columna *column1* de la tabla temporal declarada creada en el ejemplo anterior, utilice la sentencia siguiente:

```
SELECT column1 FROM SESSION.TT1;
```

Observe que DB2 también le permite crear tablas permanentes con el esquema SESSION. Si crea una tabla permanente con el nombre calificado SESSION.TT3, luego puede crear una tabla temporal declarada con el nombre calificado SESSION.TT3. En esta situación, DB2 siempre resuelve las referencias a las tablas permanente y temporal declarada con nombres calificados idénticos a la tabla temporal declarada. Para evitar confundir tablas permanentes con tablas temporales declaradas, no debe crear tablas permanentes con el esquema SESSION.

Si crea una aplicación que incluye una referencia de SQL estático a una tabla, vista o alias calificado con el esquema SESSION, el precompilador de DB2 no compila dicha sentencia en el momento de la vinculación y marca la sentencia para indicar que “necesita compilación”. En el tiempo de ejecución, DB2 compila la sentencia. Este comportamiento se denomina *vinculación incremental*. DB2 realiza automáticamente una vinculación incremental para referencias de SQL estático a tablas, vistas y alias calificados con el esquema SESSION. No hace falta que especifique la opción VALIDATE RUN en el mandato BIND o PRECOMPILE para permitir una vinculación incremental para dichas sentencias.

Si emite una sentencia ROLLBACK para una transacción que incluye una sentencia DECLARE GLOBAL TEMPORARY TABLE, DB2 elimina la tabla temporal declarada. Si emite una sentencia DROP TABLE para una tabla temporal declarada, al emitir una sentencia ROLLBACK para dicha transacción sólo se restaura una tabla temporal declarada vacía. Una sentencia ROLLBACK de una sentencia DROP TABLE no restaura las filas que existían en la tabla temporal declarada.

El comportamiento por omisión de una tabla temporal declarada consiste en suprimir todas las filas de la tabla cuando el usuario confirma una transacción. Sin embargo, si quedan uno o más cursores WITH HOLD abiertos en la tabla temporal declarada, DB2 no suprime las filas de la tabla cuando el usuario confirma una transacción. Para evitar suprimir todas las filas cuando confirma una transacción, cree la tabla temporal utilizando la cláusula ON COMMIT PRESERVE ROWS en la sentencia DECLARE GLOBAL TEMPORARY TABLE.

Si modifica el contenido de una tabla temporal declarada mediante una sentencia INSERT, UPDATE o DELETE dentro de una transacción y retrotrae dicha transacción, DB2 suprime todas las filas de la tabla temporal declarada. Si intenta modificar el contenido de una tabla temporal declarada mediante una sentencia INSERT, UPDATE o DELETE y la sentencia falla, DB2 se comporta del siguiente modo:

- Si la tabla se creó sin el parámetro NOT LOGGED (es decir, la tabla se registra), sólo se retrotraen los cambios realizados por la sentencia INSERT, UPDATE o DELETE.
- Si la tabla se creó con el parámetro NOT LOGGED, DB2 suprime todas las filas de la tabla temporal declarada.

Cuando se encuentra una anomalía en un entorno de bases de datos particionadas, todas las tablas temporales declaradas existentes en la partición de base de datos que ha fallado pasan a estar inutilizable. Cualquier acceso siguiente a estas tablas temporales declaradas inutilizables devuelve un error (SQL1477N). Cuando la aplicación encuentra una tabla temporal declarada inutilizable, la aplicación puede eliminar la tabla o volverla a crear especificando la cláusula WITH REPLACE en la sentencia DECLARE GLOBAL TEMPORARY TABLE.

Las tablas temporales declaradas están sujetas a varias restricciones. Por ejemplo, no puede definir alias ni vistas para tablas temporales declaradas. No puede utilizar IMPORT y LOAD para llenar tablas temporales declaradas. Puede, aunque con restricciones, crear índices para tablas temporales declaradas. Además, puede ejecutar RUNSTATS sobre una tabla temporal declarada para actualizar las estadísticas correspondientes a la tabla temporal declarada y sus índices.

Información relacionada:

- "Sentencia DECLARE GLOBAL TEMPORARY TABLE" en la publicación *Consulta de SQL, Volumen 2*

Ejemplos relacionados:

- "tbtemp.sqc -- How to use a declared temporary table (C)"
- "TbTemp.java -- How to use Declared Temporary Table (JDBC)"

Transmisión de grandes volúmenes de datos a través de una red

Puede combinar las técnicas de procedimientos almacenados y el bloqueo de filas para mejorar significativamente el rendimiento de las aplicaciones que necesitan pasar grandes cantidades de datos a través de una red.

Las aplicaciones que pasan matrices, grandes cantidades de datos o paquetes de datos a través de la red pueden pasar los datos en bloques utilizando como mecanismo de transporte la estructura de datos SQLDA o variables del lenguaje principal. Esta técnica es sumamente potente en los lenguajes principales que soportan estructuras.

Tanto una aplicación cliente como un procedimiento de servidor pueden pasar los datos a través de la red. Los datos se pueden pasar utilizando uno de los tipos de datos siguientes:
- VARCHAR
- LONG VARCHAR
- CLOB
- BLOB

Los datos también se pueden pasar utilizando uno de los tipos de gráficos siguientes:
- VARGRAPHIC
- LONG VARGRAPHIC
- DBCLOB

Nota: Cuando utilice esta técnica, asegúrese de considerar la posibilidad de convertir los caracteres. Si está pasando datos con uno de los tipos de datos de serie de caracteres, como por ejemplo VARCHAR, LONG VARCHAR o CLOB, o tipo de datos gráficos, como por ejemplo VARGRAPHIC, LONG VARGRAPHIC o DBCLOB, y si la página de códigos de la aplicación no es la misma que la de la base de datos, los datos que no sean de tipo carácter se convertirán como si lo fueran. Para evitar la conversión de caracteres, debe pasar los datos en una variable con el tipo de datos BLOB.

Conceptos relacionados:

- "Conversión de caracteres entre páginas de códigos diferentes" en la página 650
- "Procedimientos almacenados de DB2" en la página 19

Tareas relacionadas:

- "Specifying row blocking to reduce overhead" en la publicación *Administration Guide: Performance*

Parte 8. Apéndices

Apéndice A. Sentencias de SQL soportadas

La tabla siguiente lista todas las sentencias de SQL soportadas en DB2 Universal Database para Linux, UNIX y Windows. Todas las sentencias de la columna "Sentencia de SQL" están soportadas en aplicaciones de SQL estático. Las columnas restantes muestran si las sentencias están soportadas en otros contextos de desarrollo de aplicaciones de DB2.

Tabla 93. Sentencias de SQL (DB2 Universal Database)

Sentencia de SQL	Dinámico[1]	Procesador de línea de mandatos (CLP)	Activadores[4]	Funciones definidas por el usuario (UDF) y métodos de SQL	Procedimientos de SQL
ALLOCATE CURSOR					X
ALTER { BUFFERPOOL, DATABASE PARTITION GROUP, FUNCTION, METHOD, NICKNAME, [8] PROCEDURE, SEQUENCE, SERVER, [8] TABLE, TABLESPACE, TYPE, USER MAPPING, [8] VIEW }	X				
ASSOCIATE LOCATORS					X
BEGIN DECLARE SECTION[2]					
CALL		X[7]	X	X	X
Sentencia CASE					X
CLOSE		X			X
COMMENT ON	X	X			X
COMMIT	X	X			X
SQL compuesto (incorporado)					
Sentencia compuesta					X
CONNECT (Tipo 1)		X			
CONNECT (Tipo 2)		X			
CREATE { ALIAS, BUFFERPOOL, DATABASE PARTITION GROUP, DISTINCT TYPE, EVENT MONITOR, FUNCTION, FUNCTION MAPPING,[8] INDEX, INDEX EXTENSION, METHOD, NICKNAME,[8] PROCEDURE, SCHEMA, SEQUENCE, SERVER, TABLE, TABLESPACE, TRANSFORM, TRIGGER, TYPE, TYPE MAPPING,[8] USER MAPPING,[8] VIEW, WRAPPER[8] }	X	X			X[9]
DECLARE CURSOR[2]		X			X
DECLARE GLOBAL TEMPORARY TABLE	X	X			X
DELETE	X	X	X[3]	X	X
DESCRIBE[6]		X			

Tabla 93. Sentencias de SQL (DB2 Universal Database) (continuación)

Sentencia de SQL	Dinámico[1]	Procesador de línea de mandatos (CLP)	Activadores[4]	Funciones definidas por el usuario (UDF) y métodos de SQL	Procedimientos de SQL
DISCONNECT		X			
DROP	X	X			X[9]
END DECLARE SECTION[2]					
EXECUTE					X
EXECUTE IMMEDIATE					X
EXPLAIN	X	X			X
FETCH		X			X
FLUSH EVENT MONITOR	X	X			
FLUSH PACKAGE CACHE	X	X			X
sentencia FOR			X	X	X
FREE LOCATOR					
GET DIAGNOSTICS			X	X	X
Sentencia GOTO					X
GRANT	X	X			X
Sentencia IF			X	X	X
INCLUDE[2]					
INSERT	X	X	X[3]	X	X
ITERATE			X	X	X
Sentencia LEAVE			X	X	X
LOCK TABLE	X	X			X
Sentencia LOOP					X
MERGE	X	X	X[3]	X	X
OPEN		X			X
PREPARE					X
REFRESH TABLE	X	X			
RELEASE		X			
RELEASE SAVEPOINT	X	X			X
RENAME TABLE	X	X			
RENAME TABLESPACE	X	X			
Sentencia REPEAT					X
Sentencia RESIGNAL					X
Sentencia RETURN				X	X
REVOKE	X	X			
ROLLBACK	X	X			X
SAVEPOINT	X	X			X
Sentencia-select	X	X	X	X	

Tabla 93. Sentencias de SQL (DB2 Universal Database) (continuación)

Sentencia de SQL	Dinámico[1]	Procesador de línea de mandatos (CLP)	Activadores[4]	Funciones definidas por el usuario (UDF) y métodos de SQL	Procedimientos de SQL
SELECT INTO					X
SET CONNECTION		X			
SET CURRENT DEFAULT TRANSFORM GROUP	X	X			X
SET CURRENT DEGREE	X	X			X
SET CURRENT EXPLAIN MODE	X	X			X
SET CURRENT EXPLAIN SNAPSHOT	X	X			X
SET CURRENT ISOLATION	X	X			X
SET CURRENT LOCK TIMEOUT	X	X			X
SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION	X	X			X
SET CURRENT PACKAGE PATH					
SET CURRENT PACKAGESET					
SET CURRENT QUERY OPTIMIZATION	X	X			X
SET CURRENT REFRESH AGE	X	X			X
SET ENCRYPTION PASSWORD	X	X			X
SET EVENT MONITOR STATE	X	X			X
SET INTEGRITY	X	X			
SET PASSTHRU[8]	X	X			
SET PATH	X	X			X
SET SCHEMA	X	X			X
SET SERVER OPTION[8]	X	X			
SET SESSION AUTHORIZATION[8]	X	X			
Definir variable			X	X	X
Sentencia SIGNAL			X	X	X
SIGNAL SQLSTATE[5]	X	X			
UPDATE	X	X	X[3]	X	X
VALUES INTO					X
WHENEVER[2]					
Sentencia WHILE			X	X	X

Tabla 93. Sentencias de SQL (DB2 Universal Database) (continuación)

Sentencia de SQL	Dinámico[1]	Procesador de línea de mandatos (CLP)	Activadores[4]	Funciones definidas por el usuario (UDF) y métodos de SQL	Procedimientos de SQL

Notas:

1. Puede codificar todas las sentencias de esta lista como SQL estático, pero sólo las marcadas con X como SQL dinámico.
2. No puede ejecutar esta sentencia.
3. No puede modificar datos de tabla en activadores anteriores. Por tanto, no puede invocar procedimientos definidos con MODIFIES SQL DATA ni utilizar sentencias INSERT, UPDATE, DELETE o MERGE en activadores anteriores.
4. Las sentencias de SQL contenidas en activadores no pueden hacer referencia a variables de transición indefinidas, objetos federados ni a tablas temporales declaradas. Además, las sentencias de SQL contenidas en activadores anteriores no pueden hacer referencia a tablas de consulta materializadas que estén definidas con REFRESH IMMEDIATE.

   Para obtener una lista completa de las restricciones existentes para activadores, consulte el tema "Sentencia CREATE TRIGGER" en el manual *Consulta de SQL, Volumen 2* .
5. Solo puede utilizar esta sentencia con sentencias CREATE FUNCTION, CREATE METHOD, CREATE PROCEDURE o CREATE TRIGGER.
6. La sentencia DESCRIBE de SQL tiene una sintaxis diferente que la del mandato DESCRIBE de CLP.
7. Cuando se emite CALL desde el procesador de línea de mandatos, sólo ciertos procedimientos y sus respectivos parámetros reciben soporte.
8. La sentencia sólo recibe soporte para servidores de bases de datos federados.
9. Los procedimientos de SQL sólo pueden emitir las sentencias CREATE y DROP para índices, tablas y vistas.

Información relacionada:

- "Sentencia DESCRIBE" en la publicación *Consulta de SQL, Volumen 2*
- "Administración de los archivos JAR en el servidor de bases de datos" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor*

Apéndice B. Limitaciones en el despliegue de los plug-ins de seguridad

Existen las limitaciones siguientes en el uso de los plug-ins de seguridad:

Limitaciones de soporte del Controlador JDBC universal de DB2:

El Controlador JDBC universal de DB2® no da soporte al modelo de autenticación por plug-in en el extremo cliente. Por tanto, no podrá utilizar un plug-in de autenticación de GSS-API para conectar con un servidor de DB2 Universal Database (DB2 UDB) para Linux, UNIX®, y Windows® desde un cliente del Controlador JDBC universal de DB2. Un cliente del Controlador JDBC universal de DB2 solo puede utilizar el mecanismo de autenticación soportado a nivel del sistema operativo o el mecanismo de autenticación Kerberos. Esta limitación es aplicable tanto a la conectividad de Tipo 2 como de Tipo 4.

En concreto, no se puede asignar el valor GSSPLUGIN al parámetro de configuración *srvcon_auth* del gestor de bases de datos si al mismo tiempo el valor del parámetro de configuración *srvcon_gssplugin_list* de gestor de bases de datos no contiene el nombre de un plug-in de GSS-API basado en Kerberos.

En cambio, el parámetro *srvcon_auth* puede tener cualquiera de los valores siguientes: CLIENT, SERVER, SERVER_ENCRYPT, KERBEROS, KRB_SERVER_ENCRYPT, GSS_SERVER_ENCRYPT, DATA_ENCRYPT o DATA_ENCRYPT_CMP.

Limitaciones de soporte de la familia de productos DB2 UDB:

No puede utilizar un plug-in de GSS-API para autenticar conexiones establecidas entre un cliente de DB2 UDB para Linux, UNIX y Windows y otro servidor de la familia de productos DB2 UDB. Tampoco puede autenticar conexiones establecidas entre otro servidor de la familia de productos DB2 UDB, que actúa como cliente, y un servidor de DB2 UDB para Linux, UNIX y Windows.

Si utiliza un cliente de DB2 UDB para Linux, UNIX y Windows para conectar con otros servidores de la familia de productos DB2 UDB, puede, sin embargo, utilizar los plug-ins de ID de usuario/contraseña del extremo cliente, tal como el plug-in de autenticación del sistema operativo proporcionado por IBM®, o bien puede escribir su propio plug-in de ID de usuario/contraseña. Puede también utilizar plug-ins Kerberos o implementar su propio plug-in Kerberos.

Cuando utiliza un cliente de DB2 UDB para Linux, UNIX y Windows, no debe catalogar una base de datos utilizando el tipo de autenticación GSSPLUGIN.

Limitaciones de soporte de DB2 Information Integrator:

DB2 II no da soporte a la utilización de credenciales delegadas procedentes de un plug-in de GSS_API para establecer conexiones de salida con fuentes de datos. Las conexiones con fuentes de datos deben seguir utilizando el mandato CREATE USER MAPPING.

Limitaciones de soporte del Servidor de administración de bases de datos:

El Servidor de administración de DB2 (DAS) no da soporte a los plug-ins de seguridad. DAS solo da soporte al mecanismo de autenticación del sistema operativo.

Apéndice C. Programación en un entorno de sistema principal o iSeries

Aplicaciones en entornos de sistema principal o iSeries 739
Lenguaje de definición de datos en entornos de sistema principal e iSeries 740
Lenguaje de manipulación de datos en entornos de sistema principal e iSeries 741
Lenguaje de control de datos en entornos de sistema principal e iSeries 742
Gestión de conexiones de bases de datos con DB2 Connect 742
Proceso de peticiones de interrupción 743
Atributos de paquete, PREP y BIND. 743
Diferencias de atributos de paquete entre sistemas de bases de datos relacionales de IBM . 743
Opción CNULREQD BIND para series C terminadas en nulo 744
Variables SQLCODE y SQLSTATE autónomas 744
Niveles de aislamiento soportados por DB2 Connect 744
Órdenes de clasificación definidos por el usuario 745
Diferencias en la integridad referencial entre sistemas de bases de datos relacionales de IBM . . 745
Bloqueo y portabilidad de aplicaciones 746
Diferencias en SQLCODE y SQLSTATE entre sistemas de bases de datos relacionales de IBM . . 746
Diferencias en el catálogo del sistema entre sistemas de bases de datos relacionales de IBM . . 747
Desbordamientos por conversión numérica en asignaciones de recuperación 747
Procedimientos almacenados en entornos de sistema principal o iSeries 747
Soporte de DB2 Connect de SQL compuesto . . . 748
Actualización múltiple con DB2 Connect 749
Sentencias de SQL de servidor de sistema principal e iSeries soportadas por DB2 Connect 750
Sentencias de SQL de servidor de sistema principal e iSeries rechazadas por DB2 Connect 750

Aplicaciones en entornos de sistema principal o iSeries

DB2® Connect permite que un programa de aplicación acceda a datos de bases de datos DB2 en servidores System/390®, zSeries® e iSeries™. Por ejemplo, una aplicación que se ejecute en Windows® puede acceder a datos de una base de datos DB2 Universal Database para z/OS y OS/390. Puede crear nuevas aplicaciones o modificar las aplicaciones existentes para que se ejecuten en un entorno de sistema principal o iSeries. También es posible desarrollar aplicaciones en un entorno y trasladarlas a otro.

DB2 Connect™ le permite utilizar las API siguientes con productos de base de datos de sistema principal, como por ejemplo DB2 Universal Database para z/OS y OS/390, siempre que dichos productos soporten el elemento:

- SQL incorporado, tanto estático como dinámico
- La Interfaz a nivel de llamada de DB2
- La API ODBC de Microsoft®
- JDBC

Algunas sentencias de SQL difieren según los productos de base de datos relacional. Se puede encontrar con sentencias de SQL que:

- Sean iguales para todos los productos de base de datos que utilice, independientemente de los estándares.
- Estén disponibles en todos los productos de bases de datos relacionales de IBM® (vea la información de consulta de SQL para obtener más detalles).
- Sean exclusivas para el sistema de base de datos al que acceda.

La portabilidad de las sentencias de SQL de las dos primeras categorías es muy grande, pero las de la tercera categoría habrá que cambiarlas antes. En general, la portabilidad de las sentencias de SQL en Lenguaje de definición de datos (Data

Definition Language - DDL) no es tan grande como la de las que están en Lenguaje de manipulación de datos (Data Manipulation Language - DML).

DB2 Connect acepta algunas sentencias de SQL que DB2 Universal Database no soporta. DB2 Connect pasa estas sentencias al servidor de sistema principal o iSeries. Para obtener información sobre los límites en las distintas plataformas, como por ejemplo la longitud máxima de columna, consulte el tema sobre límites de SQL.

Si traslada una aplicación CICS® desde OS/390® o VSE para que se ejecute bajo otro producto CICS (por ejemplo, CICS para AIX), ésta también puede acceder a la base de datos OS/390 o VSE utilizando DB2 Connect. Consulte los manuales *CICS/6000 Application Programming Guide* y *CICS Customization and Operation* para obtener más detalles.

Nota: Puede utilizar DB2 Connect con una base de datos DB2 Universal Database Versión 8, aunque sólo necesita un cliente DB2. La mayoría de puntos de incompatibilidad relacionados en los temas siguientes no se producirán si utiliza DB2 Connect para una base de datos DB2 Universal Database Versión 8, excepto en aquellos casos en que exista una restricción debida a una limitación del propio DB2 Connect.

Tareas relacionadas:

- "Creación de la base de datos de ejemplo en servidores de sistema principal o AS/400 e iSeries" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Información relacionada:

- "Límites de SQL" en la publicación *Consulta de SQL, Volumen 1*

Lenguaje de definición de datos en entornos de sistema principal e iSeries

Las sentencias DDL difieren según el producto de base de datos IBM®, puesto que el almacenamiento se maneja de forma diferente en los distintos sistemas. En los sistemas de servidor de sistema principal o iSeries™, pueden existir varios pasos entre el diseño de una base de datos y la emisión de una sentencia CREATE TABLE. Por ejemplo, una serie de sentencias pueden convertir el diseño de objetos lógicos en la representación física de dichos objetos en el almacenamiento.

El precompilador pasa muchas de estas sentencias DDL al servidor de sistema principal o iSeries cuando el usuario precompila para una base de datos de servidor de sistema principal o iSeries. Las mismas sentencias no se precompilarían para una base de datos del sistema en que se esté ejecutando la aplicación. Por ejemplo, en una aplicación Windows® la sentencia CREATE STORGROUP se precompilaría satisfactoriamente para una base de datos DB2 Universal Database para z/OS y OS/390 pero no para una base de datos DB2® para Windows.

Lenguaje de manipulación de datos en entornos de sistema principal e iSeries

En general, la portabilidad de las sentencias de DML es muy grande. Las sentencias SELECT, INSERT, UPDATE y DELETE son parecidas en los distintos productos de base de datos relacional de IBM®. La mayoría de aplicaciones utilizan, principalmente, sentencias de SQL en DML que el programa DB2® Connect soporta.

A continuación se muestran algunas consideraciones que se deben tener en cuenta cuando se utilice DML en entornos de sistema principal e iSeries™:

- Tipos de datos numéricos

  Cuando se transfieran datos numéricos a DB2 Universal Database, el tipo de datos puede cambiar. Los valores SQLTYPE de tipo numérico y decimal con zona, soportados por OS/400®, se convierten en valores SQLTYPE decimales fijos (empaquetados).

- Datos de bytes mixtos

  Los datos de bytes mixtos pueden consistir en caracteres de un juego de caracteres de código UNIX® ampliado (EUC), un juego de caracteres de doble byte (DBCS) y un juego de caracteres de un solo byte (SBCS) en la misma columna. En los sistemas que almacenan los datos en EBCDIC (OS/390, z/OS™, OS/400, VSE y VM), los caracteres de cambio a teclado ideográfico y de cambio a teclado estándar marcan el comienzo y el final de los datos de doble byte. En los sistemas que almacenan los datos en ASCII (como UNIX), no se requieren caracteres de desplazamiento a teclado ideográfico ni de desplazamiento a teclado estándar.

  Si la aplicación transfiere datos de bytes mixtos desde un sistema ASCII a un sistema EBCDIC, asegúrese de que haya suficiente espacio para los caracteres de desplazamiento de teclado. Para cada cambio de datos SBCS a DBCS, añada 2 bytes a la longitud de los datos. Para mejorar la portabilidad, utilice series de longitud variable en las aplicaciones que usen datos de bytes mixtos.

- Campos largos

  Los campos largos (series que contienen más de 254 caracteres) se manejan de forma distinta en los diferentes sistemas. Un servidor de sistema principal o iSeries sólo puede soportar un subconjunto de funciones escalares para los campos largos; por ejemplo, DB2 Universal Database para z/OS y OS/390 sólo admite las funciones **LENGTH** y **SUBSTR** para los campos largos. Asimismo, un servidor de sistema principal o iSeries puede necesitar un manejo distinto de determinadas sentencias de SQL; por ejemplo, DB2 Server para VSE y VM requiere que con la sentencia INSERT sólo se utilice una variable del lenguaje principal, SQLDA, o un valor NULL.

- Tipo de datos de objetos grandes

  DB2 Connect soporta el tipo de datos LOB.

- Tipos definidos por el usuario

  DB2 Connect sólo soporta los tipos diferenciados definidos por el usuario. Los tipos estructurados, también denominados tipos de datos abstractos, no reciben soporte de DB2 Connect.

- Tipo de datos ROWID

  DB2 Connect maneja el tipo de datos ROWID como VARCHAR para datos de bits.

- Tipo de datos BIGINT

DB2 Connect soporta enteros de ocho bytes (64 bits). El tipo de datos interno BIGINT se utiliza para proporcionar soporte de cardinalidad de bases de datos muy grandes y mantener a la vez la precisión de los datos.

Lenguaje de control de datos en entornos de sistema principal e iSeries

Cada sistema de gestión de bases de datos relacionales de IBM® proporciona distintos niveles de granularidad para las sentencias GRANT y REVOKE de SQL. Para verificar las sentencias de SQL adecuadas a utilizar para cada sistema de gestión de bases de datos, consulte las publicaciones específicas del producto.

Gestión de conexiones de bases de datos con DB2 Connect

DB2® Connect soporta las versiones CONNECT TO y CONNECT RESET de la sentencia CONNECT, así como CONNECT sin ningún parámetro. Si una aplicación llama a una sentencia de SQL sin antes ejecutar una sentencia CONNECT TO explícita, se realiza una conexión *implícita* del servidor de aplicaciones por omisión (si hay alguno definido).

Cuando se conecta con una base de datos, en el campo SQLERRP de la SQLCA se devuelve información que identifica el sistema de gestión de bases de datos relacionales. Si el servidor de aplicaciones es una base de datos relacional de IBM®, los tres primeros bytes de SQLERRP contienen uno de los valores siguientes:

DSN DB2 Universal Database para z/OS y OS/390

ARI DB2 Server para VSE y VM

QSQ DB2 UDB para iSeries™

SQL DB2 Universal Database.

Si emite una sentencia CONNECT TO o CONNECT nula mientras se utiliza DB2 Connect™, el código de territorio o símbolo de territorio del campo SQLERRMC de la SQLCA se devuelve en forma de blancos; el CCSID del servidor de aplicaciones se devuelve en el símbolo de página de códigos o de juego de códigos.

Puede realizar explícitamente una desconexión mediante la sentencia CONNECT RESET (para la conexión de tipo 1), las sentencias RELEASE y COMMIT (para la conexión de tipo 2) o la sentencia DISCONNECT (para cualquier tipo de conexión, siempre que el entorno no sea de supervisor de TP).

Nota: Una aplicación puede recibir valores SQLCODE que denoten errores y la aplicación seguir terminando normalmente; en este caso, DB2 Connect confirma los datos. Si no desea que se confirmen, debe emitir un mandato ROLLBACK.

El mandato FORCE permite desconectar de la base de datos usuarios seleccionados o todos los usuarios. Este mandato se soporta para bases de datos de servidor de sistema principal e iSeries; el usuario puede imponer una desconexión de la estación de trabajo DB2 Connect.

Información relacionada:

- "Sentencia CONNECT (Tipo 1)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CONNECT (Tipo 2)" en la publicación *Consulta de SQL, Volumen 2*

Proceso de peticiones de interrupción

DB2® Connect maneja una petición de interrupción procedente de un cliente DB2 de una de estas dos formas:

- Si aparece la palabra clave INTERRUPT_ENABLED en el campo PARMS de la entrada del catálogo DCS, DB2 Connect™ eliminará la conexión con el servidor de sistema principal o iSeries™ cuando reciba una petición de interrupción. La pérdida de conexión, al menos en servidores DB2 UDB para OS/390® y z/OS™, hará que se interrumpa la petición actual en el servidor.
- Si no aparece la palabra clave INTERRUPT_ENABLED en el campo PARMS de la entrada del catálogo DCS, las peticiones de interrupción se pasan por alto.

Atributos de paquete, PREP y BIND

Las secciones siguientes describen las diferencias en los atributos de paquete entre los sistemas de bases de datos relacionales de IBM y las consideraciones a tener en cuenta al utilizar los mandatos PREPCOMPILE y BIND.

Diferencias de atributos de paquete entre sistemas de bases de datos relacionales de IBM

Un paquete tiene los atributos siguientes:

ID de colección
ID del paquete. Se puede especificar en el mandato PREP.

Propietario
ID de autorización del propietario del paquete. Se puede especificar en los mandatos PREP o BIND.

Creador
Nombre del usuario que vincula el paquete.

Calificador
Calificador implícito de los objetos del paquete. Se puede especificar en los mandatos PREP o BIND.

Cada sistema de servidor de sistema principal o iSeries™ tiene limitaciones en el uso de estos atributos:

DB2 Universal Database para z/OS y OS/390
Los cuatro atributos pueden ser diferentes. El uso de un calificador distinto requiere privilegios de administración especiales. Para obtener más información sobre las condiciones relativas al uso de estos atributos, consulte la publicación *Command Reference* para DB2 Universal Database para z/OS y OS/390.

DB2 Server para VSE y VM
Todos los atributos deben ser idénticos. Si USER1 crea un archivo de vinculación (mediante PREP) y USER2 realiza la vinculación real, USER2 necesitará autorización DBA para realizar la vinculación para USER1. Para los atributos sólo se utiliza el nombre de usuario de USER1.

DB2® UDB para iSeries
El calificador indica el nombre de colección. La relación entre calificadores y propietario afecta al otorgamiento y a la revocación de privilegios sobre el objeto. El nombre de usuario que se registra es el creador y propietario,

a menos que se califique con un ID de colección, en cuyo caso el ID de colección es el propietario. El ID de colección debe existir antes de que se pueda utilizar como calificador.

DB2 Universal Database
Los cuatro atributos pueden ser diferentes. El uso de un propietario distinto requiere autorización de administración y el enlazador debe tener el privilegio CREATEIN sobre el esquema (si ya existe).

Opción CNULREQD BIND para series C terminadas en nulo

La opción de vinculación CNULREQD altera temporalmente el manejo de las series terminadas en nulo que se especifican utilizando la opción LANGLEVEL.

Por omisión, CNULREQD se establece en YES. Esto hace que las series terminadas en nulo se interpreten según los estándares MIA. Si se conecta a un servidor DB2 Universal Database para z/OS y OS/390, es altamente recomendable establecer CNULREQD en YES. Es necesario que las aplicaciones codificadas según los estándares SAA1 (con respecto a las series de caracteres terminadas en nulo) se vinculen con la opción CNULREQD establecida en NO. De lo contrario, las series terminadas en nulo se interpretarán según los estándares MIA, incluso si se preparan utilizando LANGLEVEL establecido en SAA1.

Conceptos relacionados:

- "Series terminadas en nulo en C y C++" en la página 169

Variables SQLCODE y SQLSTATE autónomas

Las variables SQLCODE y SQLSTATE autónomas, definidas en ISO/ANS SQL92, se soportan mediante la opción de precompilación LANGLEVEL SQL92E. Durante la precompilación se emitirá un aviso SQL0020W, indicando que no se soporta LANGLEVEL. Este aviso sólo es aplicable a las características relacionadas bajo el título LANGLEVEL MIA, que es un subconjunto de LANGLEVEL SQL92E.

Información relacionada:

- "Mandato PRECOMPILE" en la publicación *Consulta de mandatos*

Niveles de aislamiento soportados por DB2 Connect

DB2 Connect acepta los niveles de aislamiento siguientes cuando se prepara o vincula una aplicación:

RR Lectura repetible

RS Estabilidad de lectura

CS Estabilidad del cursor

UR Lectura no confirmada

NC Sin confirmación

Los niveles de aislamiento se relacionan en orden de mayor a menor protección. Si el servidor de sistema principal o iSeries™ no soporta el nivel de aislamiento que se especifique, se utiliza el próximo nivel soportado superior.

La tabla siguiente muestra el resultado de cada nivel de aislamiento en cada servidor de aplicaciones de sistema principal o iSeries.

Tabla 94. Niveles de aislamiento

DB2 Connect	DB2 Universal Database para z/OS y OS/390	DB2 Server para VSE y VM	DB2® UDB para iSeries	DB2 Universal Database
RR	RR	RR	nota 1	RR
RS	nota 2	RR	COMMIT(*ALL)	RS
CS	CS	CS	COMMIT(*CS)	CS
UR	nota 3	CS	COMMIT(*CHG)	UR
NC	nota 4	nota 5	COMMIT(*NONE)	UR

Notas:

1. En DB2 UDB no existe ninguna opción COMMIT equivalente que coincida con RR. DB2 UDB para iSeries da soporte a RR bloqueando la tabla entera.
2. Resultados en RR para la Versión 3.1, y resultados en RS para la Versión 4.1 con el APAR PN75407 o para la Versión 5.1.
3. Resultados en CS para la Versión 3.1, y resultados en UR para la Versión 4.1 o para la Versión 5.1.
4. Resultados en CS para la Versión 3.1, y resultados en UR para la Versión 4.1 con el APAR PN60988 o para la Versión 5.1.
5. El nivel de aislamiento NC no se soporta con DB2 Server para VSE y VM.

Con DB2 UDB, puede acceder a una tabla que no sea de diario si una aplicación se enlaza con un nivel de aislamiento de UR y el bloqueo establecido en ALL, o si el nivel de aislamiento se establece en NC.

Órdenes de clasificación definidos por el usuario

Las diferencias entre EBCDIC y ASCII ocasionan diferencias en los órdenes de clasificación en los diversos productos de base de datos, y también afectan a las cláusulas ORDER BY y GROUP BY. Una manera de minimizar estas diferencias consiste en crear un orden de clasificación definido por el usuario que imite el orden de clasificación EBCDIC. Sólo se puede especificar un orden de clasificación cuando se crea una base de datos nueva.

Nota: Ahora, las tablas de bases de datos se pueden almacenar en formato ASCII de DB2 Universal Database para z/OS y OS/390. Esto permite un intercambio más rápido de los datos entre DB2 Connect y DB2 Universal Database para z/OS y OS/390, y suprime la necesidad de proporcionar procedimientos de campo, que en lugar de esto se deberían utilizar para convertir los datos y volverlos a colocar en secuencia.

Diferencias en la integridad referencial entre sistemas de bases de datos relacionales de IBM

Los diferentes sistemas manejan las restricciones referenciales de forma distinta:

DB2 Universal Database para z/OS y OS/390
Para poder crear una clave foránea utilizando la clave primaria, antes se debe crear un índice sobre la clave primaria. Las tablas se pueden hacer referencia a sí mismas.

DB2 Server para VSE y VM
Se crea automáticamente un índice para una clave foránea. Las tablas no se pueden hacer referencia a sí mismas.

DB2® UDB para iSeries™
Se crea automáticamente un índice para una clave foránea. Las tablas se pueden hacer referencia a sí mismas.

DB2 Universal Database
Para las bases de datos DB2 Universal Database, se crea automáticamente un índice para una restricción exclusiva, incluida una clave primaria. Las tablas se pueden hacer referencia a sí mismas.

Existen otras normas que varían con respecto a los niveles de cascada.

Bloqueo y portabilidad de aplicaciones

La manera en que el servidor de bases de datos realiza un bloqueo puede afectar a algunas aplicaciones. Por ejemplo, las aplicaciones diseñadas en base a un bloqueo de fila y aislamiento de la estabilidad del cursor no se pueden trasladar directamente a sistemas que realicen un bloqueo de página. Debido a estas diferencias fundamentales, es posible que haya que ajustar las aplicaciones.

Los productos DB2 Universal Database para z/OS y OS/390 y DB2 Universal Database ofrecen la posibilidad de marcar un tiempo de espera para un bloqueo y enviar un código de retorno de error a las aplicaciones que están a la espera.

Diferencias en SQLCODE y SQLSTATE entre sistemas de bases de datos relacionales de IBM

Distintos productos de bases de datos relacionales de IBM® no siempre producen los mismos valores SQLCODE para errores similares. Puede tratar este problema de una de dos maneras:

- Utilice el SQLSTATE en lugar del SQLCODE para un error determinado.

  Los valores SQLSTATE tienen, aproximadamente, el mismo significado en los diversos productos de base de datos, y los productos producen valores SQLSTATE que corresponden a los valores SQLCODE.
- Correlacione los valores SQLCODE de un sistema con los de otro sistema.

  Por omisión, DB2® Connect correlaciona los valores SQLCODE y los símbolos de cada sistema servidor de sistema principal IBM o iSeries™ con su sistema DB2 Universal Database. El usuario puede especificar su propio archivo de correlación de SQLCODE si desea alterar temporalmente la correlación por omisión o está utilizando un servidor de bases de datos que no dispone de correlación de SQLCODE (un servidor de bases de datos no IBM). También puede desactivar la correlación de SQLCODE.

Conceptos relacionados:

- "SQLCODE mapping" en la publicación *DB2 Connect User's Guide*

Diferencias en el catálogo del sistema entre sistemas de bases de datos relacionales de IBM

Los catálogos del sistema varían en los distintos productos de base de datos de IBM®. Muchas de las diferencias se pueden enmascarar mediante el uso de vistas. Para obtener información, consulte le documentación correspondiente al servidor de bases de datos que utilice.

Las funciones de catálogo de la CLI evitan este problema presentando soporte de la misma API y conjuntos de resultados para las consultas de catálogos en la familia DB2®.

Conceptos relacionados:

- "Catalog functions for querying system catalog information in CLI applications" en la publicación *CLI Guide and Reference, Volume 1*

Desbordamientos por conversión numérica en asignaciones de recuperación

Los desbordamientos por conversión numérica en asignaciones de recuperación pueden ser gestionados de forma diferente por los diversos productos de bases de datos relaciones de IBM®. Por ejemplo, considere la operación de traer una columna de coma flotante a una variable del lenguaje principal de entero desde DB2 Universal Database para z/OS y OS/390 y desde DB2 Universal Database. Cuando se convierte el valor de coma flotante en un valor entero, se puede producir un desbordamiento por conversión. Por omisión, DB2 Universal Database para z/OS y OS/390 devolverá a la aplicación un SQLCODE de aviso y un valor nulo. En cambio, DB2 Universal Database devolverá un error de desbordamiento por conversión. Es recomendable que las aplicaciones eviten los desbordamientos por conversión numérica en las asignaciones de recuperación y realicen el movimiento a variables del lenguaje principal del tamaño adecuado.

Procedimientos almacenados en entornos de sistema principal o iSeries

Las consideraciones que deben tenerse en cuenta para procedimientos almacenados en entornos de sistema principal e iSeries™ son las siguientes:

- Invocación

  Un programa cliente puede invocar a un programa servidor emitiendo una sentencia CALL de SQL. En este caso, cada servidor funciona de un modo distinto a los otros servidores.

  z/OS™ y OS/390®
  : El nombre de esquema no debe tener una longitud superior a 8 bytes, el nombre de procedimiento no debe contener más de 18 bytes y el procedimiento almacenado debe estar definido en el catálogo SYSIBM.SYSPROCEDURES del servidor.

  VSE o VM
  : El nombre de procedimiento no debe contener más de 18 bytes y debe estar definido en el catálogo SYSTEM.SYSROUTINES del servidor.

OS/400®

El nombre de procedimiento debe ser un identificador de SQL. También puede utilizar las sentencias DECLARE PROCEDURE o CREATE PROCEDURE para especificar el nombre de vía de acceso real (nombre de esquema o nombre de colección) a fin de localizar el procedimiento almacenado.

Todas las sentencias CALL destinadas a DB2® UDB para iSeries desde REXX/SQL se deben preparar dinámicamente y la aplicación las tiene que ejecutar como la sentencia CALL implantada en las correlaciones de REXX/SQL con CALL USING DESCRIPTOR.

Puede invocar al programa servidor en DB2 Universal Database con el mismo convenio de parámetros que utilizan los programas servidores en DB2 Universal Database para z/OS y OS/390, DB2 UDB para iSeries o DB2 Server para VSE y VM. Para obtener más información sobre el convenio de parámetros en otras plataformas, consulte la documentación del producto DB2 correspondiente a cada plataforma.

Todas las sentencias de SQL contenidas en un procedimiento almacenado se ejecutan formando parte de la unidad de trabajo SQL iniciada por el programa cliente SQL.

- No pase valores de indicador con significados especiales a los procedimientos almacenados, ni desde éstos.

  Entre DB2 Universal Database, los sistemas pasan cualquier cosa que se incluya en las variables indicadoras. Sin embargo, cuando se utiliza DB2 Connect™, sólo se pueden pasar los valores 0, -1 y -128 en las variables indicadoras.
- Debe definir un parámetro para devolver los posibles errores o avisos encontrados por la aplicación de servidor.

  Un programa servidor en DB2 Universal Database puede actualizar la SQLCA para devolver los posibles errores o avisos, pero un procedimiento almacenado en DB2 Universal Database para z/OS y OS/390 o DB2 UDB para iSeries no dispone de este soporte. Si desea devolver un código de error desde el procedimiento almacenado, lo debe pasar en forma de parámetro. El SQLCODE y la SQLCA sólo los establece el servidor para los errores detectados por el sistema.
- DB2 Server para VSE y VM Versión 7 o superior, DB2 Universal Database para z/OS y OS/390 Versión 5.1 o superior, DB2 para AS/400® V5R1 y DB2 para iSeries Versión 7 o superior son los únicos servidores de aplicaciones de sistema principal o iSeries que actualmente pueden devolver los conjuntos de resultados de procedimientos almacenados.

Conceptos relacionados:

- "Procedimientos almacenados de DB2" en la página 19

Información relacionada:

- "Sentencia CALL" en la publicación *Consulta de SQL, Volumen 2*

Soporte de DB2 Connect de SQL compuesto

El código SQL compuesto permite que se agrupen varias sentencias de SQL en un solo bloque ejecutable. Esto puede reducir la actividad general de la red y mejorar el tiempo de respuesta.

Con SQL compuesto NOT ATOMIC, el proceso de SQL compuesto continúa después de un error. Con SQL compuesto ATOMIC, un error retrotrae el grupo entero de SQL compuesto.

Las sentencias se seguirán ejecutando hasta que el servidor de aplicaciones las termine. En general, la ejecución de la sentencia de SQL compuesto sólo se detendrá en caso de errores graves.

El código SQL compuesto NOT ATOMIC se puede utilizar con todos los servidores de aplicación de sistema principal o iSeries™ soportados. SQL compuesto ATOMIC se puede utilizar con servidores de aplicaciones de sistema principal soportados.

Si se producen varios errores de SQL, los valores SQLSTATE de las siete primeras sentencias anómalas se devuelven en el campo SQLERRMC de la SQLCA, con un mensaje que indica que se han producido varios errores.

Información relacionada:

- "SQLCA" en la publicación *Administrative API Reference*

Actualización múltiple con DB2 Connect

DB2® Connect permite realizar una actualización múltiple, también conocida como confirmación en dos fases. Una actualización múltiple es una actualización de varias bases de datos en una sola unidad de trabajo distribuida (DUOW). Si se puede o no se puede utilizar esta posibilidad, depende de varios factores:

- El programa de aplicación se debe precompilar con las opciones CONNECT 2 y SYNCPOINT TWOPHASE.
- Si tiene conexiones de red SNA, puede utilizar el soporte de confirmación en dos fases que proporciona la función del gestor de puntos de sincronismo (SPM) de DB2 Connect™ Enterprise Edition en AIX® y Windows® NT. Esta función permite a los siguientes servidores de bases de datos de sistema principal participar en una unidad de trabajo distribuida:
  - DB2 para AS/400® Versión 3.1 o posterior
  - DB2 UDB para iSeries™ Versión 5.1 o posterior
  - DB2 para OS/390® Versión 5.1 o posterior
  - DB2 UDB para OS/390 y z/OS™ Versión 7 o posterior
  - DB2 para VM & VSE Versión V5.1 o posterior.

  Lo anterior es cierto para aplicaciones DB2 UDB nativas y aplicaciones coordinadas por un supervisor de TP externo tal como IBM® TXSeries®, CICS® para Open Systems, BEA Tuxedo, Encina® Monitor y Microsoft® Transaction Server.
- Si tiene conexiones de red TCP/IP, un servidor DB2 para OS/390 V5.1 o posteriores puede participar en una unidad de trabajo distribuida. Si la aplicación se controla mediante un Supervisor de proceso de transacciones, tal como IBM TXSeries, CICS para Open Systems, Encina Monitor o Microsoft Transaction Server, debe utilizar SPM.

  Si, tanto las aplicaciones DB2 nativas como las aplicaciones de supervisor de TP, utilizan un servidor DB2 Connect Enterprise Edition común para acceder a datos del sistema principal a través de conexiones TCP/IP, se debe utilizar el gestor de puntos de sincronismo.

  Si se utiliza un solo servidor DB2 Connect Enterprise Edition para acceder a datos del sistema principal usando los dos protocolos de red SNA y TCP/IP, y si

se requiere una confirmación en dos fases, debe utilizar SPM. Esto es así tanto para las aplicaciones DB2 como para las aplicaciones de supervisor de TP.

Conceptos relacionados:

- "XA function supported by DB2 Universal Database" en la publicación *Administration Guide: Planning*
- "Configuring DB2 Connect with an XA compliant transaction manager" en la publicación *DB2 Connect User's Guide*

Tareas relacionadas:

- "Configuring BEA Tuxedo" en la publicación *Administration Guide: Planning*
- "Updating host or iSeries database servers with an XA-compliant transaction manager" en la publicación *Administration Guide: Planning*

Sentencias de SQL de servidor de sistema principal e iSeries soportadas por DB2 Connect

Las sentencias siguientes se compilan satisfactoriamente para un proceso de servidor de sistema principal o iSeries™, pero no para un proceso con sistemas DB2 Universal Database:

- ACQUIRE
- DECLARE (modificador.(calificador.)nombre_tabla TABLE ...
- LABEL ON

El procesador de línea de mandatos también soporta estas sentencias.

Las sentencias siguientes se soportan para un proceso de servidor de sistema principal e iSeries, pero no se añaden al archivo de vinculación ni al paquete y el procesador de línea de mandatos no las soporta:

- DESCRIBE nombre_sentencia INTO nombre_descriptor USING NAMES
- PREPARE nombre_sentencia INTO nombre_descriptor USING NAMES FROM ...

El precompilador realiza las suposiciones siguientes:

- Las variables del lenguaje principal son variables de entrada
- Se asigna a la sentencia un número de sección exclusivo.

Sentencias de SQL de servidor de sistema principal e iSeries rechazadas por DB2 Connect

DB2® Connect y el procesador de línea de mandatos no soportan las sentencias de SQL siguientes:

- COMMIT WORK RELEASE
- DECLARE nombre_sentencia, nombre_sentencia STATEMENT
- DESCRIBE nombre_sentencia INTO nombre_descriptor USING xxxx (donde xxxx es ANY, BOTH o LABELS)
- PREPARE nombre_sentencia INTO nombre_descriptor USING xxxx FROM :variable_sist_principal (donde xxxx es ANY, BOTH o LABELS)
- PUT ...
- ROLLBACK WORK RELEASE

- SET :host_variable = CURRENT ...

Las sentencias de SQL dinámicas ampliadas de DB2 Server para VSE y VM se rechazan con los valores -104 y SQLCODE de error de sintaxis.

Apéndice D. Simulación de clasificación binaria EBCDIC

Con DB2®, puede clasificar series de caracteres de acuerdo con un orden de clasificación que haya definido. Puede utilizar esta función para simular la clasificación binaria EBCDIC.

Como ejemplo de cómo simular una clasificación EBCDIC, supongamos que desea crear una base de datos ASCII con la página de códigos 850, pero también desea que las series de caracteres se clasifiquen como si los datos realmente residieran en una base de datos EBCDIC con la página de códigos 500. Consulte las figuras siguientes para ver las definiciones de página de códigos 500 y página de códigos 850.

Supongamos la clasificación relativa de cuatro caracteres en una base de datos de página de códigos EBCDIC 500, cuando se clasifican en código binario:

Carácter	**Punto de código de página de códigos 500**
'a'	X'81'
'b'	X'82'
'A'	X'C1'
'B'	X'C2'

El orden de clasificación binaria de la página de códigos 500 (el orden deseado) es:

```
'a' < 'b' < 'A' < 'B'
```

Si crea la base de datos con la página de códigos ASCII 850, el orden de clasificación sería el siguiente:

Carácter	**Punto de código de página de códigos 850**
'a'	X'61'
'b'	X'62'
'A'	X'41'
'B'	X'42'

El orden binario de la página de códigos 850 (que no es el orden deseado) es:

```
'A' < 'B' < 'a' < 'b'
```

Para conseguir el orden deseado, tiene que crear la base de datos con un orden de clasificación definido por el usuario. Con ese fin se proporciona un orden de clasificación de ejemplo junto con DB2 en el archivo de inclusión `sqle850a.h`. El contenido de `sqle850a.h` se muestra a continuación.

```
#ifndef SQL_H_SQLE850A
#define SQL_H_SQLE850A

#ifdef __cplusplus
extern "C" {
#endif

unsigned char sqle_850_500[256] = {
0x00,0x01,0x02,0x03,0x37,0x2d,0x2e,0x2f,0x16,0x05,0x25,0x0b,0x0c,0x0d,0x0e,0x0f,
0x10,0x11,0x12,0x13,0x3c,0x3d,0x32,0x26,0x18,0x19,0x3f,0x27,0x1c,0x1d,0x1e,0x1f,
0x40,0x4f,0x7f,0x7b,0x5b,0x6c,0x50,0x7d,0x4d,0x5d,0x5c,0x4e,0x6b,0x60,0x4b,0x61,
0xf0,0xf1,0xf2,0xf3,0xf4,0xf5,0xf6,0xf7,0xf8,0xf9,0x7a,0x5e,0x4c,0x7e,0x6e,0x6f,
0x7c, 0xc1 , 0xc2 ,0xc3,0xc4,0xc5,0xc6,0xc7,0xc8,0xc9,0xd1,0xd2,0xd3,0xd4,0xd5,0xd6,
0xd7,0xd8,0xd9,0xe2,0xe3,0xe4,0xe5,0xe6,0xe7,0xe8,0xe9,0x4a,0xe0,0x5a,0x5f,0x6d,
0x79, 0x81 , 0x82 ,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x91,0x92,0x93,0x94,0x95,0x96,
0x97,0x98,0x99,0xa2,0xa3,0xa4,0xa5,0xa6,0xa7,0xa8,0xa9,0xc0,0xbb,0xd0,0xa1,0x07,
0x68,0xdc,0x51,0x42,0x43,0x44,0x47,0x48,0x52,0x53,0x54,0x57,0x56,0x58,0x63,0x67,
0x71,0x9c,0x9e,0xcb,0xcc,0xcd,0xdb,0xdd,0xdf,0xec,0xfc,0x70,0xb1,0x80,0xbf,0xff,
0x45,0x55,0xce,0xde,0x49,0x69,0x9a,0x9b,0xab,0xaf,0xba,0xb8,0xb7,0xaa,0x8a,0x8b,
0x2b,0x2c,0x09,0x21,0x28,0x65,0x62,0x64,0xb4,0x38,0x31,0x34,0x33,0xb0,0xb2,0x24,
0x22,0x17,0x29,0x06,0x20,0x2a,0x46,0x66,0x1a,0x35,0x08,0x39,0x36,0x30,0x3a,0x9f,
0x8c,0xac,0x72,0x73,0x74,0x0a,0x75,0x76,0x77,0x23,0x15,0x14,0x04,0x6a,0x78,0x3b,
0xee,0x59,0xeb,0xed,0xcf,0xef,0xa0,0x8e,0xae,0xfe,0xfb,0xfd,0x8d,0xad,0xbc,0xbe,
0xca,0x8f,0x1b,0xb9,0xb6,0xb5,0xe1,0x9d,0x90,0xbd,0xb3,0xda,0xfa,0xea,0x3e,0x41
};
#ifdef __cplusplus
}
#endif

#endif /* SQL_H_SQLE850A */
```

Figura 66. Orden de clasificación definido por el usuario - sqle_850_500

Para ver cómo conseguir el orden de clasificación de la página de códigos 500 en caracteres de la página de códigos 850, observe el orden de clasificación de ejemplo de `sqle_850_500`. Para cada carácter de la página de códigos 850, su peso en el orden de clasificación es simplemente su punto de código correspondiente en la página de códigos 500.

Por ejemplo, supongamos que tenemos la letra 'a'. Esta letra tiene el punto de código X'61' para la página de códigos 850. En la matriz `sqle_850_500`, a la letra 'a' se le asigna un peso de X'81' (es decir, el elemento número 98 de la matriz `sqle_850_500`).

Veamos cómo se clasifican los cuatro caracteres cuando se crea la base de datos con el orden de clasificación definido por el usuario del ejemplo anterior:

Carácter	Punto código página códigos 850 / Peso (de sqle_850_500)
'a'	X'61' / X'81'
'b'	X'62' / X'82'
'A'	X'41' / X'C1'
'B'	X'42' / X'C2'

El orden definido por el usuario de la página de códigos 850 por peso (el orden deseado) es:

```
'a' < 'b' < 'A' < 'B'
```

En este ejemplo, consigue el orden deseado especificando los pesos correctos para simular el comportamiento deseado.

Mirando detenidamente el orden de clasificación real, se observa que el orden en sí es simplemente una tabla de conversión, en la que la página de códigos fuente es la página de códigos de la base de datos (850) y la página de códigos de destino es la página de códigos del orden binario deseado (500). Otros órdenes de clasificación de ejemplo que se suministran con DB2 permiten otras conversiones. Si una tabla de conversión que necesita no se suministra con DB2, puede obtener tablas de conversión adicionales de la publicación de IBM® *Character Data Representation Architecture, Reference and Registry*, SC09-2190. Encontrará las tablas de conversión adicionales en un CD-ROM que se suministra con dicha publicación.

Dígitos hex 1 → 2 ↓	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0	(SP) SP010000	& SM030000	- SP010000	ø LO610000	Ø LO620000	° SM190000	µ SM170000	¢ SC040000	{ SM110000	} SM140000	\ SM070000	0 ND100000
-1	(RSP) SP300000	é LE110000	/ SP120000	É LE120000	a LA010000	j LJ010000	~ SD190000	£ SC020000	A LA020000	J LJ020000	÷ SA060000	1 ND010000
-2	â LA150000	ê LE150000	Â LA160000	Ê LE160000	b LB010000	k LK010000	s LS010000	¥ SC050000	B LB020000	K LK020000	S LS020000	2 ND020000
-3	ä LA170000	ë LE170000	Ä LA180000	Ë LE180000	c LC010000	l LL010000	t LT010000	· SD630000	C LC020000	L LL020000	T LT020000	3 ND030000
-4	à LA130000	è LE130000	À LA140000	È LE140000	d LD010000	m LM010000	u LU010000	© SM520000	D LD020000	M LM020000	U LU020000	4 ND040000
-5	á LA110000	í LI110000	Á LA120000	Í LI120000	e LE010000	n LN010000	v LV010000	§ SM240000	E LE020000	N LN020000	V LV020000	5 ND050000
-6	ã LA190000	î LI150000	Ã LA200000	Î LI160000	f LF010000	o LO010000	w LW010000	¶ SM250000	F LF020000	O LO020000	W LW020000	6 ND060000
-7	å LA270000	ï LI170000	Å LA280000	Ï LI180000	g LG010000	p LP010000	x LX010000	¼ NF040000	G LG020000	P LP020000	X LX020000	7 ND070000
-8	ç LC410000	ì LI130000	Ç LC420000	Ì LI140000	h LH010000	q LQ010000	y LY010000	½ NF010000	H LH020000	Q LQ020000	Y LY020000	8 ND080000
-9	ñ LN190000	ß LS610000	Ñ LN200000	` SD130000	i LI010000	r LR010000	z LZ010000	¾ NF050000	I LI020000	R LR020000	Z LZ020000	9 ND090000
-A	[SM060000	] SM080000	¦ SM650000	: SP130000	« SP170000	ª SM210000	¡ SP030000	¬ SM660000	(SHY) SP320000	1 ND011000	2 ND021000	3 ND031000
-B	. SP110000	$ SC030000	, SP080000	# SM010000	» SP180000	º SM200000	¿ SP160000	Ì SM130000	ô LO150000	û LU150000	Ô LO160000	Û LU160000
-C	< SA030000	* SM040000	% SM020000	@ SM050000	ð LD630000	æ LA510000	Ð LD620000	¯ SM150000	ö LO170000	ü LU170000	Ö LO180000	Ü LU180000
-D	(SP060000	) SP070000	_ SP090000	' SP050000	ý LY110000	, SD410000	Ý LY120000	¨ SD170000	ò LO130000	ù LU130000	Ò LO140000	Ù LU140000
-E	+ SA010000	; SP140000	> SA050000	= SA040000	þ LT630000	Æ LA520000	Þ LT640000	´ SD110000	ó LO110000	ú LU110000	Ó LO120000	Ú LU120000
-F	! SP020000	^ SD150000	? SP150000	" SP040000	± SA020000	¤ SC010000	® SM530000	× SA070000	õ LO190000	ÿ LY170000	Õ LO200000	(EO)

Página de códigos 00500

Figura 67. Página de códigos 500

Dígitos hex 1 → 2 ↓	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0		► SM590000	(SP) SP010000	0 ND100000	@ SM050000	P LP020000	` SD130000	p LP010000	Ç LC420000	É LE120000	á LA110000	▒ SF140000	┌ SF020000	ð LD630000	Ó LO120000	(SH̄Y) SP320000
-1	☺ SS000000	◄ SM630000	! SP020000	1 ND010000	A LA020000	Q LQ020000	a LA010000	q LQ010000	ü LU170000	æ LA510000	í LI110000	▓ SF150000	┴ SF070000	Đ LD620000	ß LS610000	± SA020000
-2	☻ SS010000	↕ SM760000	" SP040000	2 ND020000	B LB020000	R LR020000	b LB010000	r LR010000	é LE110000	Æ LA520000	ó LO110000	╫ SF160000	┬ SF060000	Ê LE160000	Ô LO160000	═ SM100000
-3	♥ SS020000	!! SP330000	# SM010000	3 ND030000	C LC020000	S LS020000	c LC010000	s LS010000	â LA150000	ô LO150000	ú LU110000	┤ SF110000	├ SF080000	Ë LE180000	Ò LO140000	¾ NF050000
-4	♦ SS030000	¶ SM250000	$ SC030000	4 ND040000	D LD020000	T LT020000	d LD010000	t LT010000	ä LA170000	ö LO170000	ñ LN190000	╪ SF090000	┼ SF100000	È LE140000	õ LO190000	¶ SM250000
-5	♣ SS040000	§ SM240000	% SM020000	5 ND050000	E LE020000	U LU020000	e LE010000	u LU010000	à LA130000	ò LO130000	Ñ LN200000	Á LA120000	╬ SF050000	1 LI610000	Õ LO200000	§ SM240000
-6	♠ SS050000	▬ SM700000	& SM030000	6 ND060000	F LF020000	V LV020000	f LF010000	v LV010000	å LA270000	û LU150000	ª SM210000	Â LA160000	ã LA190000	Í LI120000	µ SM170000	÷ SA060000
-7	• SM570000	↨ SM770000	' SP050000	7 ND070000	G LG020000	W LW020000	g LG010000	w LW010000	ç LC410000	ù LU130000	º SM200000	À LA140000	Ã LA200000	Î LI160000	þ LT630000	, SD410000
-8	◘ SM570001	↑ SM320000	(SP060000	8 ND080000	H LH020000	X LX020000	h LH010000	x LX010000	ê LE150000	ÿ LY170000	¿ SP160000	© SM520000	╘ SF380000	Ï LI180000	Þ LT640000	° SM190000
-9	○ SM750000	↓ SM330000	) SP070000	9 ND090000	I LI020000	Y LY020000	i LI010000	y LY010000	ë LE170000	Ö LO180000	® SM530000	╒ SF230000	╓ SF390000	╔ SF040000	Ú LU120000	¨ SD170000
-A	◙ SM750002	→ SM310000	* SM040000	: SP130000	J LJ020000	Z LZ020000	j LJ010000	z LZ010000	è LE130000	Ü LU180000	¬ SM660000	╕ SF240000	╙ SF400000	╗ SF010000	Û LU160000	• SD630000
-B	♂ SM280000	← SM300000	+ SA010000	; SP140000	K LK020000	[SM060000	k LK010000	{ SM110000	ï LI170000	ø LO610000	½ NF010000	╣ SF250000	╫ SF410000	▀ SF610000	Ù LU140000	1 ND011000
-C	♀ SM290000	└ SA420000	, SP080000	< SA030000	L LL020000	\ SM070000	l LL010000	l SM130000	î LI150000	£ SC020000	¼ NF040000	╝ SF260000	╪ SF420000	▄ SP570000	ý LY110000	3 ND031000
-D	♪ SM930000	↔ SM780000	- SP100000	= SA040000	M LM020000	] LM080000	m LM010000	} SM140000	ì LI130000	Ø LO620000	¡ SP030000	¢ SC040000	╤ SF430000	¦ SM650000	Ý LY120000	2 ND021000
-E	♫ SM910000	▲ SM600000	. SP110000	> SA050000	N LN020000	^ SD150000	n LN010000	~ SD190000	Ä LA180000	× SA070000	« SP170000	¥ SC050000	╬ SF440000	Ì LI140000	¯ SM150000	■ SM470000
-F	☼ SM690000	▼ SV040000	/ SP120000	? SP150000	O LO020000	_ SP090000	o LO010000	⌂ SM790000	Å LA280000	ƒ SC070000	» SP180000	╖ SF030000	¤ SC010000	▬ SF600000	´ SD110000	(RSP) SP300000

Página de códigos 00850

Figura 68. Página de códigos 850

Conceptos relacionados:

- "Ordenes de clasificación" en la página 637

Información relacionada:

- "sqlecrea - Create Database" en la publicación *Administrative API Reference*

Apéndice E. Información técnica sobre DB2 Universal Database

Documentación y ayuda de DB2

Está disponible información técnica de DB2® a través de las herramientas y los métodos siguientes:

- Centro de información de DB2
  - Temas
  - Herramientas de ayuda para DB2
  - Programas de ejemplo
  - Guías de aprendizaje
- Archivos PDF descargables y en CD y manuales impresos
  - Guías
  - Manuales de consulta
- Ayuda de línea de mandatos
  - Ayuda de mandatos
  - Ayuda de mensajes
  - Ayuda para estados de SQL
- Código fuente instalado
  - Programas de ejemplo

Puede acceder a información técnica adicional de DB2 Universal Database™ como, por ejemplo, notas técnicas, white papers y Redbooks™ en línea en ibm.com®. Acceda al sitio de la biblioteca de software de gestión de información de DB2 en www.ibm.com/software/data/pubs/.

Actualizaciones de la documentación de DB2

De forma periódica, IBM® puede realizar FixPaks de la documentación y otras actualizaciones de la misma en el Centro de información de DB2 disponible. Si accede al Centro de información de DB2 en http://publib.boulder.ibm.com/infocenter/db2help/, siempre visualizará la información más actualizada. Si ha instalado el Centro de información de DB2 localmente, tendrá que instalar cualquier actualización de forma manual para poder visualizarla. Las actualizaciones de la documentación le permiten actualizar la información que ha instalado desde el *CD del Centro de información de DB2* cuando está disponible nueva información.

El Centro de información se actualiza con mayor frecuencia que los manuales PDF o en copia impresa. Para conseguir la información técnica de DB2 más actualizada, instale las actualizaciones de la documentación a medida que estén disponibles o diríjase al Centro de información de DB2 en el sitio www.ibm.com.

Conceptos relacionados:

- "CLI sample programs" en la publicación *CLI Guide and Reference, Volume 1*
- "Programas de ejemplo Java" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Centro de información de DB2" en la página 758

Tareas relacionadas:

- "Invocación de ayuda según contexto desde una herramienta de DB2" en la página 776
- "Actualización del Centro de información de DB2 instalado en el sistema o en un servidor de intranet" en la página 768
- "Invocación de la ayuda de mensajes desde el procesador de línea de mandatos" en la página 778
- "Invocación de la ayuda de mandatos desde el procesador de línea de mandatos" en la página 778
- "Invocación de la ayuda para estados de SQL desde el procesador de línea de mandatos" en la página 779

Información relacionada:

- "Documentación PDF e impresa de DB2" en la página 770

Centro de información de DB2

El Centro de información de DB2® le proporciona acceso a toda la información que necesita para obtener el máximo provecho de los productos de la familia de DB2, incluidos DB2 Universal Database™, DB2 Connect™, DB2 Information Integrator y DB2 Query Patroller™. El Centro de información de DB2 también contiene información relativa a las características y los componentes principales de DB2, como la duplicación, el depósito de datos y DB2 Extenders.

El Centro de información de DB2 presenta las características siguientes si se visualiza en Mozilla 1.0 o posterior o bien en Microsoft® Internet Explorer 5.5 o posterior. Algunas características requieren que se habilite el soporte de JavaScript™:

Opciones flexibles de instalación
Puede elegir visualizar la documentación de DB2 utilizando la opción que mejor se ajuste a sus necesidades:

- Para asegurarse fácilmente de que la documentación siempre esté actualizada, puede acceder a toda la documentación directamente desde el Centro de información de DB2 incluido en el sitio Web de IBM® de http://publib.boulder.ibm.com/infocenter/db2help/
- Para minimizar el esfuerzo de actualización y mantener el tráfico de red en su intranet, puede instalar la documentación de DB2 en un solo servidor de la intranet
- Para maximizar la flexibilidad y reducir la dependencia de las conexiones de red, puede instalar la documentación de DB2 en su propio sistema

Búsqueda
Es posible buscar en todos los temas del Centro de información de DB2 entrando un término de búsqueda en el campo de texto **Buscar**. Puede recuperar coincidencias exactas encerrando los términos entre comillas y puede afinar la búsqueda mediante operadores de comodín (*, ?) y operadores booleanos (AND, NOT, OR).

Tabla de contenido orientada a tareas
Puede localizar los temas en la documentación de DB2 a partir de una sola tabla de contenido. La tabla de contenido está organizada principalmente

según la clase de tareas que puede desear realizar, pero también incluye entradas para visiones generales de productos, objetivos, información de consulta, un índice y un glosario.

- Las visiones generales de los productos describen la relación entre los productos disponibles en la familia de DB2, las características que ofrece cada uno de estos productos y proporcionan información actualizada del release de cada uno de estos productos.
- Las categorías de objetivos, como la instalación, la administración y el desarrollo, incluyen temas que permiten realizar rápidamente tareas y desarrollar un conocimiento más profundo de la información de fondo para realizar dichas tareas.
- Los temas de consulta proporcionan información detallada sobre un tema, incluida la sintaxis de sentencias y mandatos, la ayuda de mensajes y los parámetros de configuración.

Mostrar el tema actual en la tabla de contenido
Puede mostrar dónde encaja el tema actual en la tabla de contenido pulsando el botón **Renovar / Mostrar tema actual** en el marco de la tabla de contenido o pulsando el botón **Mostrar en tabla de contenido** en el marco del contenido. Esta característica es útil si ha seguido varios enlaces con temas relacionados en varios archivos o ha llegado a un tema a partir de resultados de una búsqueda.

Índice Es posible acceder a toda la documentación desde el índice. El índice está organizado en orden alfabético por términos del índice.

Glosario
Puede utilizar el glosario a fin de buscar definiciones de términos utilizados en la documentación de DB2. El glosario está organizado en orden alfabético por términos del glosario.

Información adaptada integrada
El Centro de información de DB2 visualiza la información en el idioma preferido que se ha establecido en las preferencias de navegador. Si un tema no está disponible en el idioma preferido del usuario, el Centro de información de DB2 visualiza la versión inglesa de ese tema.

Si desea información técnica sobre iSeries™, consulte el centro de información de IBM eServer™ iSeries en www.ibm.com/eserver/iseries/infocenter/.

Conceptos relacionados:

- "Escenarios de instalación del Centro de información de DB2" en la página 760

Tareas relacionadas:

- "Actualización del Centro de información de DB2 instalado en el sistema o en un servidor de intranet" en la página 768
- "Visualización de temas en el idioma preferido en el Centro de información de DB2" en la página 769
- "Invocación del Centro de información de DB2" en la página 767
- "Instalación del Centro de información de DB2 utilizando el asistente de instalación de DB2 (UNIX)" en la página 762
- "Instalación del Centro de información de DB2 utilizando el asistente de instalación de DB2 (Windows)" en la página 765

Escenarios de instalación del Centro de información de DB2

Los entornos de trabajo distintos pueden plantear requisitos distintos para el modo de acceder a la información de DB2®. Se puede acceder al Centro de información de DB2 en el sitio Web de IBM®, en un servidor de la red de la organización o en una versión instalada en el sistema. En los tres casos, la documentación está incluida en el Centro de información de DB2, el cual consiste en una Web estructurada de información que se organiza en temas y que se visualiza mediante un navegador. Por omisión, los productos de DB2 acceden al Centro de información de DB2 en el sitio Web de IBM. No obstante, si desea acceder al Centro de información de DB2 en un servidor de intranet o en su propio sistema, es necesario que instale el Centro de información de DB2 utilizando el CD del Centro de información de DB2 que encontrará en el Paquete de soportes del producto. Consulte el siguiente resumen de opciones para acceder a la documentación de DB2, junto con los tres escenarios de instalación, como ayuda para determinar qué método de acceso al Centro de información de DB2 le funciona mejor en su entorno de trabajo y qué cuestiones relacionadas con la instalación se pueden tener en cuenta.

Resumen de opciones para acceder a la documentación de DB2:

La siguiente tabla proporciona recomendaciones sobre las opciones que son posibles en su entorno de trabajo a la hora de acceder a la documentación de productos de DB2 del Centro de información de DB2.

Acceso a Internet	Acceso a Intranet	Recomendación
Sí	Sí	Acceda al Centro de información de DB2 en el sitio Web de IBM o acceda al Centro de información de DB2 instalado en un servidor de intranet.
Sí	No	Acceda al Centro de información de DB2 en el sitio Web de IBM.
No	Sí	Acceda al Centro de información de DB2 instalado en un servidor de intranet.
No	No	Acceda al Centro de información de DB2 en un sistema local.

Escenario: Acceso al Centro de información de DB2 en su sistema:

Tsu-Chen es propietario de una fábrica en una pequeña ciudad que no dispone de ISP local para proporcionarle acceso a Internet. Ha adquirido DB2 Universal Database™ para la gestión de su inventario, pedidos de productos, información de cuentas bancarias y gastos empresariales. Puesto que nunca había utilizado un producto de DB2 anteriormente, Tsu-Chen tendrá que aprender a partir de la documentación de productos de DB2.

Después de instalar DB2 Universal Database en el sistema utilizando la opción de instalación típica, Tsu-Chen intenta acceder a la documentación de DB2. Sin embargo, el navegador emite un mensaje de error que indica que la página que ha intentado abrir no se encuentra. Tsu-Chen comprueba el manual de instalación de su producto de DB2 y descubre que tiene que instalar el Centro de información de DB2 si desea acceder a la documentación de DB2 en su sistema. Encuentra el *CD del Centro de información de DB2* en el paquete de soportes y lo instala.

Desde el programa ejecutor de aplicaciones del sistema operativo, Tsu-Chen dispone ahora de acceso al Centro de información de DB2 y puede aprender a utilizar el producto de DB2 para incrementar el éxito de su empresa.

Escenario: Acceso al Centro de información de DB2 en el sitio Web de IBM:

Colin es un consultor de tecnologías de la información con una empresa de formación. Está especializado en tecnología de bases de datos y SQL y ofrece clases sobre estos temas a empresas por toda Norteamérica utilizando DB2 Universal Database. Parte de las clases de Colin incluye el uso de la documentación de DB2 como una herramienta didáctica. Por ejemplo, mientras imparte los cursos sobre SQL, Colin utiliza la documentación de DB2 relativa a SQL como un modo de enseñar sintaxis básica y avanzada para las consultas de base de datos.

La mayoría de las empresas en las que Colin imparte cursos tienen acceso a Internet. Esta situación ha influido en la decisión de Colin de configurar su sistema portátil para que acceda al Centro de información de DB2 en el sitio Web de IBM cuando ha instalado la versión más reciente de DB2 Universal Database. Dicha configuración permite a Colin disponer de acceso en línea a la documentación más reciente de DB2 durante sus clases.

Sin embargo, a veces, mientras viaja, Colin no tiene acceso a Internet. Esto le planteaba un problema, especialmente cuando necesitaba acceder a la documentación de DB2 para preparar las clases. A fin de evitar tales situaciones, Colin ha instalado una copia del Centro de información de DB2 en el sistema portátil.

Colin disfruta de la flexibilidad que supone tener siempre una copia de la documentación de DB2 a su disposición. Mediante el mandato **db2set**, puede configurar fácilmente las variables de registro en el sistema portátil para acceder al Centro de información de DB2 en el sitio Web de IBM o en el sistema portátil, según su situación.

Escenario: Acceso al Centro de información de DB2 en un servidor de intranet:

El trabajo de Eva es el de administrador sénior de bases de datos en una compañía de seguros de vida. Sus responsabilidades administrativas incluyen la instalación y configuración de la versión más reciente de DB2 Universal Database en los servidores de bases de datos UNIX® de la compañía. Recientemente, la compañía ha informado a sus empleados de que, por razones de seguridad, no se les proporcionará acceso a Internet en el trabajo. Dado que la compañía tiene un entorno de red, Eva decide instalar una copia del Centro de información de DB2 en un servidor de intranet a fin de que todos los empleados de la compañía que utilicen el depósito de datos de la misma de forma regular (representantes de ventas, gestores de ventas y analistas de empresa) tengan acceso a la documentación de DB2.

Eva indica a su equipo encargado de las bases de datos que instalen la versión más reciente de DB2 Universal Database en los sistemas de todos los empleados a través de un archivo de respuestas, para asegurarse de que cada sistema esté configurado de manera que acceda al Centro de información de DB2 utilizando el nombre de sistema principal y el número de puerto del servidor de intranet.

No obstante, debido a un malentendido, Miguel, un administrador de bases de datos auxiliar del equipo de Eva, instala una copia del Centro de información de DB2 en varios sistemas de los empleados en lugar de configurar DB2 Universal

Database para que acceda al Centro de información de DB2 en el servidor de intranet. Con el fin de corregir esta situación, Eva indica a Miguel que utilice el mandato **db2set** para cambiar las variables de registro del Centro de información de DB2 (DB2_DOCHOST para el nombre de sistema principal y DB2_DOCPORT para el número de puerto) en cada uno de esos sistemas. Ahora todos los sistemas correspondientes de la red tienen acceso al Centro de información de DB2, y los empleados pueden hallar las respuestas a sus preguntas sobre DB2 en la documentación de DB2.

Conceptos relacionados:

- "Centro de información de DB2" en la página 758

Tareas relacionadas:

- "Actualización del Centro de información de DB2 instalado en el sistema o en un servidor de intranet" en la página 768
- "Instalación del Centro de información de DB2 utilizando el asistente de instalación de DB2 (UNIX)" en la página 762
- "Instalación del Centro de información de DB2 utilizando el asistente de instalación de DB2 (Windows)" en la página 765
- "Establecimiento de la ubicación para acceder al Centro de información de DB2"

Información relacionada:

- "db2set - Mandato Registro de perfiles de DB2" en la publicación *Consulta de mandatos*

Instalación del Centro de información de DB2 utilizando el asistente de instalación de DB2 (UNIX)

Se puede acceder a la documentación de los productos de DB2 de tres maneras: en el sitio Web de IBM, en un servidor de intranet o en una versión instalada en el sistema. Por omisión, el acceso de los productos de DB2 dentro de la documentación de DB2 se efectúa en el sitio Web de IBM. Si desea acceder a la documentación de DB2 en un servidor de intranet o en su propio sistema, deberá instalar la documentación desde el *CD del Centro de información de DB2*. Mediante el asistente de instalación de DB2, puede definir sus preferencias de instalación e instalar el Centro de información de DB2 en un sistema que utilice un sistema operativo UNIX.

Prerrequisitos:

Este apartado lista los requisitos de hardware, sistema operativo, software y comunicaciones para instalar el Centro de información de DB2 en los sistemas UNIX.

- **Requisitos de hardware**

  Necesita uno de los procesadores siguientes:

  – PowerPC (AIX)
  – HP 9000 (HP-UX)
  – Intel de 32 bits (Linux)
  – Sistemas Solaris UltraSPARC (Entorno operativo Solaris)
- **Requisitos de sistema operativo**

  Necesita uno de los sistemas operativos siguientes:

- IBM AIX 5.1 (en PowerPC)
- HP-UX 11i (en HP 9000)
- Red Hat Linux 8.0 (en Intel de 32 bits)
- SuSE Linux 8.1 (en Intel de 32 bits)
- Sun Solaris Versión 8 (en sistemas UltraSPARC del Entorno operativo Solaris)

Nota: El Centro de información de DB2 se ejecuta en un subconjunto de los sistemas operativos UNIX en los que están soportados los clientes DB2. Por consiguiente, es recomendable que acceda al Centro de información de DB2 desde el sitio Web de IBM o que instale el Centro de información de DB2 y acceda al mismo en un servidor de intranet.

- **Requisitos de software**
  - Está soportado el navegador siguiente:
    - Mozilla Versión 1.0 o superior
- El asistente de instalación de DB2 es un instalador gráfico. Debe disponer de una implementación del software X Window System capaz de representar una interfaz gráfica de usuario para que el asistente de instalación de DB2 se ejecute en el sistema. A fin de ejecutar el asistente de instalación de DB2, debe asegurarse de que ha exportado debidamente la visualización. Por ejemplo, entre el mandato siguiente en el indicador de mandatos:

  ```
  export DISPLAY=9.26.163.144:0.
  ```

- **Requisitos de comunicaciones**
  - TCP/IP

Procedimiento:

Para instalar el Centro de información de DB2 utilizando el asistente de instalación de DB2:

1. Inicie una sesión en el sistema.
2. Inserte y monte el CD del producto Centro de información de DB2 en el sistema.
3. Vaya al directorio en el que está montado el CD entrando el mandato siguiente:

   ```
   cd /cd
   ```

   donde */cd* representa el punto de montaje del CD.
4. Entre el mandato **./db2setup** para iniciar el asistente de instalación de DB2.
5. Se abrirá el Área de ejecución para la instalación de IBM DB2. Para continuar directamente con la instalación del Centro de información de DB2, pulse en **Instalar producto**. Existe ayuda en línea disponible para guiarle durante los pasos restantes. Para invocar la ayuda en línea, pulse en **Ayuda**. Puede pulsar en **Cancelar** en cualquier momento para interrumpir la instalación.
6. En la página **Seleccione el producto que desee instalar**, pulse en **Siguiente**.
7. Pulse en **Siguiente** en la página **Bienvenido al asistente de instalación de DB2**. El asistente de instalación de DB2 le guiará durante el proceso de instalación del programa.
8. Para continuar con la instalación, debe aceptar el contrato de licencia. En la página **Contrato de licencia**, seleccione **Acepto los términos del contrato de licencia** y pulse en **Siguiente**.
9. Seleccione **Instalar el Centro de información de DB2 en este sistema** en la página **Seleccionar la acción de instalación**. Si desea utilizar un archivo de

respuestas para instalar el Centro de información de DB2 en éste o en otros sistemas más adelante, seleccione **Guardar los valores en un archivo de respuestas**. Pulse en **Siguiente**.

10. Seleccione los idiomas en los que se instalará el Centro de información de DB2 en la página **Seleccionar los idiomas a instalar**. Pulse en **Siguiente**.
11. Configure el Centro de información de DB2 para las comunicaciones entrantes en la página **Especificar el puerto del Centro de información de DB2**. Pulse en **Siguiente** para continuar la instalación.
12. Revise las opciones de instalación que ha elegido en la página **Comenzar a copiar archivos**. Para cambiar cualquier valor, pulse en **Anterior**. Pulse en **Instalar** para copiar los archivos del Centro de información de DB2 en el sistema.

También puede instalar el Centro de información de DB2 utilizando un archivo de respuestas.

Los archivos de anotaciones cronológicas de instalación `db2setup.his`, `db2setup.log` y `db2setup.err` están ubicados, por omisión, en el directorio `/tmp`.

El archivo `db2setup.log` capta toda la información de instalación del producto de DB2, incluidos los errores. El archivo `db2setup.his` registra todas las instalaciones de productos de DB2 en el sistema. DB2 añade el archivo `db2setup.log` al archivo `db2setup.his`. El archivo `db2setup.err` capta cualquier salida de errores devuelta por Java, como, por ejemplo, información de interrupciones y excepciones.

Cuando se haya completado la instalación, el Centro de información de DB2 estará instalado en uno de los directorios siguientes, según el sistema operativo UNIX:

- AIX: /usr/opt/db2_08_01
- HP-UX: /opt/IBM/db2/V8.1
- Linux: /opt/IBM/db2/V8.1
- Entorno operativo Solaris: /opt/IBM/db2/V8.1

Conceptos relacionados:

- "Centro de información de DB2" en la página 758
- "Escenarios de instalación del Centro de información de DB2" en la página 760

Tareas relacionadas:

- "Instalación de DB2 utilizando un archivo de respuestas (UNIX)" en la publicación *Suplemento de instalación y configuración*
- "Actualización del Centro de información de DB2 instalado en el sistema o en un servidor de intranet" en la página 768
- "Visualización de temas en el idioma preferido en el Centro de información de DB2" en la página 769
- "Invocación del Centro de información de DB2" en la página 767
- "Instalación del Centro de información de DB2 utilizando el asistente de instalación de DB2 (Windows)" en la página 765

Instalación del Centro de información de DB2 utilizando el asistente de instalación de DB2 (Windows)

Se puede acceder a la documentación de los productos de DB2 de tres maneras: en el sitio Web de IBM, en un servidor de intranet o en una versión instalada en el sistema. Por omisión, el acceso de los productos de DB2 dentro de la documentación de DB2 se efectúa en el sitio Web de IBM. Si desea acceder a la documentación de DB2 en un servidor de intranet o en su propio sistema, deberá instalar la documentación de DB2 desde el *CD del Centro de información de DB2*. Mediante el asistente de instalación de DB2, puede definir sus preferencias de instalación e instalar el Centro de información de DB2 en un sistema que utilice un sistema operativo Windows.

Prerrequisitos:

Este apartado lista los requisitos de hardware, sistema operativo, software y comunicaciones para instalar el Centro de información de DB2 en Windows.

- **Requisitos de hardware**

  Necesita uno de los procesadores siguientes:

  – Sistemas de 32 bits: una CPU Pentium o compatible con Pentium
- **Requisitos de sistema operativo**

  Necesita uno de los sistemas operativos siguientes:

  – Windows 2000

  – Windows XP

  Nota: El Centro de información de DB2 se ejecuta en un subconjunto de los sistemas operativos Windows en los que están soportados los clientes DB2. Por consiguiente, es recomendable que acceda al Centro de información de DB2 en el sitio Web de IBM o que instale el Centro de información de DB2 y acceda al mismo en un servidor de intranet.
- **Requisitos de software**

  – Están soportados los navegadores siguientes:

    - Mozilla 1.0 o superior

    - Internet Explorer Versión 5.5 ó 6.0 (Versión 6.0 para Windows XP)
- **Requisitos de comunicaciones**

  – TCP/IP

Restricciones:

- Necesita una cuenta con privilegios administrativos para instalar el Centro de información de DB2.

Procedimiento:

Para instalar el Centro de información de DB2 utilizando el asistente de instalación de DB2:

1. Inicie una sesión en el sistema con la cuenta que ha definido para la instalación del Centro de información de DB2.
2. Inserte el CD en la unidad. Si está habilitada, la característica de ejecución automática inicia el Área de ejecución para la instalación de IBM DB2.
3. El asistente de instalación de DB2 determina el idioma del sistema y ejecuta el programa de instalación para ese idioma. Si desea ejecutar el programa de

instalación en un idioma distinto del inglés o bien el programa de instalación no se inicia de forma automática, puede iniciar el asistente de instalación de DB2 manualmente.

Para iniciar el asistente de instalación de DB2 manualmente:

a. Pulse en **Inicio** y seleccione **Ejecutar**.
b. En el campo **Abrir**, escriba el mandato siguiente:

```
x:\setup.exe /i identificador de idioma de 2 letras
```

donde *x:* representa la unidad de CD, e *identificador de idioma de 2 letras* representa el idioma en el que se ejecutará el programa de instalación.

c. Pulse en **Aceptar**.

4. Se abrirá el Área de ejecución para la instalación de IBM DB2. Para continuar directamente con la instalación del Centro de información de DB2, pulse en **Instalar producto**. Existe ayuda en línea disponible para guiarle durante los pasos restantes. Para invocar la ayuda en línea, pulse en **Ayuda**. Puede pulsar en **Cancelar** en cualquier momento para interrumpir la instalación.
5. En la página **Seleccione el producto que desee instalar**, pulse en **Siguiente**.
6. Pulse en **Siguiente** en la página **Bienvenido al asistente de instalación de DB2**. El asistente de instalación de DB2 le guiará durante el proceso de instalación del programa.
7. Para continuar con la instalación, debe aceptar el contrato de licencia. En la página **Contrato de licencia**, seleccione **Acepto los términos del contrato de licencia** y pulse en **Siguiente**.
8. Seleccione **Instalar el Centro de información de DB2 en este sistema** en la página **Seleccionar la acción de instalación**. Si desea utilizar un archivo de respuestas para instalar el Centro de información de DB2 en éste o en otros sistemas más adelante, seleccione **Guardar los valores en un archivo de respuestas**. Pulse en **Siguiente**.
9. Seleccione los idiomas en los que se instalará el Centro de información de DB2 en la página **Seleccionar los idiomas a instalar**. Pulse en **Siguiente**.
10. Configure el Centro de información de DB2 para las comunicaciones entrantes en la página **Especificar el puerto del Centro de información de DB2**. Pulse en **Siguiente** para continuar la instalación.
11. Revise las opciones de instalación que ha elegido en la página **Comenzar a copiar archivos**. Para cambiar cualquier valor, pulse en **Anterior**. Pulse en **Instalar** para copiar los archivos del Centro de información de DB2 en el sistema.

Puede instalar el Centro de información de DB2 utilizando un archivo de respuestas. También es posible utilizar el mandato **db2rspgn** a fin de generar un archivo de respuestas basado en una instalación existente.

Para obtener información sobre los errores encontrados durante la instalación, consulte los archivos `db2.log` y `db2wi.log` ubicados en el directorio '`Mis documentos`'`\DB2LOG\`. La ubicación del directorio '`Mis documentos`' dependerá de la configuración de su sistema.

El archivo `db2wi.log` capta la información de la instalación de DB2 más reciente. El archivo `db2.log` capta el historial de instalaciones de productos de DB2.

Conceptos relacionados:

- "Centro de información de DB2" en la página 758

- "Escenarios de instalación del Centro de información de DB2" en la página 760

Tareas relacionadas:

- "Instalación de un producto DB2 utilizando un archivo de respuestas (Windows)" en la publicación *Suplemento de instalación y configuración*
- "Actualización del Centro de información de DB2 instalado en el sistema o en un servidor de intranet" en la página 768
- "Visualización de temas en el idioma preferido en el Centro de información de DB2" en la página 769
- "Invocación del Centro de información de DB2" en la página 767
- "Instalación del Centro de información de DB2 utilizando el asistente de instalación de DB2 (UNIX)" en la página 762

Información relacionada:

- "db2rspgn - Mandato del Generador de archivos de respuestas (Windows)" en la publicación *Consulta de mandatos*

Invocación del Centro de información de DB2

El Centro de información de DB2 proporciona acceso a toda la información que necesita para utilizar productos de DB2 para los sistemas operativos Linux, UNIX y Windows, tales como DB2 Universal Database, DB2 Connect, DB2 Information Integrator y DB2 Query Patroller.

Puede invocar el Centro de información de DB2 desde una de las ubicaciones siguientes:

- Sistemas en los que está instalado un cliente o servidor DB2 UDB
- Un servidor de intranet o sistema local en el que está instalado el Centro de información de DB2
- El sitio Web de IBM

Prerrequisitos:

Antes de invocar el Centro de información de DB2:

- *Opcional*: Configure el navegador para que visualice los temas en su idioma preferido
- *Opcional*: Configure el cliente DB2 para que utilice el Centro de información de DB2 instalado en el sistema o servidor de intranet

Procedimiento:

Para invocar el Centro de información de DB2 en un sistema en el que está instalado un cliente o servidor DB2 UDB:

- Desde el menú Inicio (sistema operativo Windows): Pulse en **Inicio** → **Programas** → **IBM DB2** → **Información** → **Centro de información**.
- Desde el indicador de línea de mandatos:
  - En los sistemas operativos Linux y UNIX, emita el mandato **db2icdocs**.
  - En el sistema operativo Windows, emita el mandato **db2icdocs.exe**.

Para abrir el Centro de información de DB2 instalado en un servidor de intranet o sistema local en un navegador Web:

- Abra la página Web en http://<nombre-sistemaprincipal>:<número-puerto>/, donde <nombre-sistemaprincipal> representa el nombre de sistema principal y <número-puerto> representa el número de puerto en el que está disponible el Centro de información de DB2.

Para abrir el Centro de información de DB2 en el sitio Web de IBM en un navegador Web:

- Abra la página Web en publib.boulder.ibm.com/infocenter/db2help/.

Conceptos relacionados:

- "Centro de información de DB2" en la página 758
- "Escenarios de instalación del Centro de información de DB2" en la página 760

Tareas relacionadas:

- "Visualización de temas en el idioma preferido en el Centro de información de DB2" en la página 769
- "Invocación de ayuda según contexto desde una herramienta de DB2" en la página 776
- "Actualización del Centro de información de DB2 instalado en el sistema o en un servidor de intranet" en la página 768
- "Invocación de la ayuda de mandatos desde el procesador de línea de mandatos" en la página 778
- "Establecimiento de la ubicación para acceder al Centro de información de DB2"

Información relacionada:

- "Mandato HELP" en la publicación *Consulta de mandatos*

Actualización del Centro de información de DB2 instalado en el sistema o en un servidor de intranet

El Centro de información de DB2 que hay disponible en http://publib.boulder.ibm.com/infocenter/db2help/ se actualizará periódicamente con documentación nueva o modificada. Asimismo, IBM puede efectuar actualizaciones del Centro de información de DB2 disponibles para descargar e instalar en el sistema o servidor de intranet. La actualización del Centro de información de DB2 no actualiza los productos de cliente o servidor DB2.

Prerrequisitos:

Es necesario tener acceso a un sistema que esté conectado a Internet.

Procedimiento:

Para actualizar el Centro de información de DB2 instalado en el sistema o servidor de intranet:

1. Abra el Centro de información de DB2 que se encuentra en el sitio Web de IBM de: http://publib.boulder.ibm.com/infocenter/db2help/
2. En la sección de descargas de la página de bienvenida, bajo la cabecera de servicio y soporte, pulse en el enlace de **documentación de DB2 Universal Database**.
3. Determine si la versión de su Centro de información de DB2 está anticuada comparando el nivel de la última imagen de documentación renovada con el

nivel de documentación que tenga instalado. El nivel de documentación que ha instalado aparece listado en la página de bienvenida del Centro de información de DB2.

4. Si se encuentra disponible una versión más reciente del Centro de información de DB2, descargue la última imagen renovada del *Centro de información de DB2* aplicable a su sistema operativo.
5. Para instalar la imagen renovada del *Centro de información de DB2*, siga las instrucciones proporcionadas en la página Web.

Conceptos relacionados:

- "Escenarios de instalación del Centro de información de DB2" en la página 760

Tareas relacionadas:

- "Invocación del Centro de información de DB2" en la página 767
- "Instalación del Centro de información de DB2 utilizando el asistente de instalación de DB2 (UNIX)" en la página 762
- "Instalación del Centro de información de DB2 utilizando el asistente de instalación de DB2 (Windows)" en la página 765

Visualización de temas en el idioma preferido en el Centro de información de DB2

El Centro de información de DB2 intenta visualizar los temas en el idioma especificado en las preferencias de navegador. Si un tema no se ha traducido al idioma preferido del usuario, el Centro de información de DB2 visualiza dicho tema en inglés.

Procedimiento:

Para visualizar temas en su idioma preferido en el navegador Internet Explorer:

1. En Internet Explorer, pulse el botón **Herramientas** —> **Opciones de Internet** —> **Idiomas...**. Se abrirá la ventana Preferencias de idioma.
2. Asegúrese de que su idioma preferido esté especificado como la primera entrada de la lista de idiomas.
   - Para añadir un nuevo idioma a la lista, pulse el botón **Agregar...**.

     Nota: La adición de un idioma no garantiza que el sistema tenga los fonts necesarios para visualizar los temas en el idioma preferido.
   - Para mover un idioma hacia el principio de la lista, seleccione el idioma y pulse el botón **Subir** hasta que el idioma esté en primer lugar en la lista de idiomas.
3. Renueve la página a fin de visualizar el Centro de información de DB2 en su idioma preferido.

Para visualizar temas en su idioma preferido en el navegador Mozilla:

1. En Mozilla, seleccione el botón **Edit** —> **Preferences** —> **Languages**. Se visualizará el panel Languages en la ventana Preferences.
2. Asegúrese de que su idioma preferido esté especificado como la primera entrada de la lista de idiomas.
   - Para añadir un nuevo idioma a la lista, pulse el botón **Add...** a fin de seleccionar un idioma en la ventana Add Languages.

- Para mover un idioma hacia el principio de la lista, seleccione el idioma y pulse el botón **Move Up** hasta que el idioma esté en primer lugar en la lista de idiomas.

3. Renueve la página a fin de visualizar el Centro de información de DB2 en su idioma preferido.

Conceptos relacionados:

- "Centro de información de DB2" en la página 758

Documentación PDF e impresa de DB2

Las tablas siguientes proporcionan los nombres oficiales de los manuales, los números de documento y los nombres de los archivos PDF. Para solicitar manuales en copia impresa, debe conocer el nombre oficial del manual. Para imprimir un archivo PDF, debe conocer el nombre del archivo PDF.

La documentación de DB2 está categorizada según las cabeceras siguientes:

- Información básica de DB2
- Información de administración
- Información para el desarrollo de aplicaciones
- Información de Business Intelligence
- Información de DB2 Connect
- Información de iniciación
- Información de aprendizaje
- Información sobre componentes opcionales
- Notas del release

Las tablas siguientes describen, para cada manual de la biblioteca de DB2, la información necesaria para solicitar la copia impresa o para imprimir o ver el PDF correspondiente al manual en cuestión. Se encuentra una descripción completa de cada uno de los manuales de la biblioteca de DB2 en el Centro de publicaciones de IBM de www.ibm.com/shop/publications/order

Información básica de DB2

La información de estos manuales es fundamental para todos los usuarios de DB2; encontrará útil esta información tanto si es programador o administrador de bases de datos como si trabaja con DB2 Connect, DB2 Warehouse Manager u otros productos de DB2.

Tabla 95. Información básica de DB2

Nombre	Número de documento	Nombre de archivo PDF
IBM DB2 Universal Database Consulta de mandatos	SC10-3725	db2n0x81
IBM DB2 Universal Database Glosario	Sin número de documento	db2t0x81
IBM DB2 Universal Database Consulta de mensajes, Volumen 1	GC10-3728, no disponible en copia impresa	db2m1x81
IBM DB2 Universal Database Consulta de mensajes, Volumen 2	GC10-3729, no disponible en copia impresa	db2m2x81
IBM DB2 Universal Database Novedades	SC10-3734	db2q0x81

Información de administración

La información de estos manuales incluye los temas necesarios para diseñar, implementar y mantener de forma efectiva bases de datos de DB2, depósitos de datos y sistemas federados.

Tabla 96. Información de administración

Nombre	Número de documento	Nombre de archivo PDF
IBM DB2 Universal Database Administration Guide: Planning	SC09-4822	db2d1x81
IBM DB2 Universal Database Administration Guide: Implementation	SC09-4820	db2d2x81
IBM DB2 Universal Database Administration Guide: Performance	SC09-4821	db2d3x81
IBM DB2 Universal Database Administrative API Reference	SC09-4824	db2b0x81
IBM DB2 Universal Database Data Movement Utilities Guide and Reference	SC09-4830	db2dmx81
IBM DB2 Universal Database Data Recovery and High Availability Guide and Reference	SC09-4831	db2hax81
IBM DB2 Universal Database Data Warehouse Center Administration Guide	SC27-1123	db2ddx81
IBM DB2 Universal Database Consulta de SQL, Volumen 1	SC10-3730	db2s1x81
IBM DB2 Universal Database Consulta de SQL, Volumen 2	SC10-3731	db2s2x81
IBM DB2 Universal Database System Monitor Guide and Reference	SC09-4847	db2f0x81

Información para el desarrollo de aplicaciones

La información de estos manuales es de especial interés para los programadores de aplicaciones o programadores que trabajan con DB2 Universal Database (DB2 UDB). Hallará información acerca de los lenguajes y compiladores soportados, así como la documentación necesaria para acceder a DB2 UDB utilizando las diversas interfaces de programación soportadas, como, por ejemplo, SQL incorporado, ODBC, JDBC, SQLJ y CLI. Si utiliza el Centro de información de DB2, también podrá acceder a versiones HTML del código fuente para los programas de ejemplo.

Tabla 97. Información para el desarrollo de aplicaciones

Nombre	Número de documento	Nombre de archivo PDF
IBM DB2 Universal Database Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones	SC10-3733	db2axx81

Tabla 97. Información para el desarrollo de aplicaciones (continuación)

Nombre	Número de documento	Nombre de archivo PDF
IBM DB2 Universal Database Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente	SC10-3723	db2a1x81
IBM DB2 Universal Database Guía de desarrollo de aplicaciones: Programación de aplicaciones de servidor	SC10-3724	db2a2x81
IBM DB2 Universal Database Call Level Interface Guide and Reference, Volume 1	SC09-4849	db2l1x81
IBM DB2 Universal Database Call Level Interface Guide and Reference, Volume 2	SC09-4850	db2l2x81
IBM DB2 Universal Database Data Warehouse Center Application Integration Guide	SC27-1124	db2adx81
IBM DB2 XML Extender Administración y programación	SC10-3750	db2sxx81

Información de Business Intelligence

La información de estos manuales describe cómo utilizar los componentes que mejoran las posibilidades de análisis y de depósito de datos de DB2 Universal Database.

Tabla 98. Información de Business Intelligence

Nombre	Número de documento	Nombre de archivo PDF
IBM DB2 Warehouse Manager Standard Edition Information Catalog Center Administration Guide	SC27-1125	db2dix81
IBM DB2 Warehouse Manager Standard Edition Installation Guide	GC27-1122	db2idx81
IBM DB2 Warehouse Manager Standard Edition Managing ETI Solution Conversion Programs with DB2 Warehouse Manager	SC18-7727	iwhe1mstx80

Información de DB2 Connect

La información incluida en esta categoría describe cómo acceder a datos de servidores de sistema principal y de sistema medio utilizando DB2 Connect Enterprise Edition o DB2 Connect Personal Edition.

Tabla 99. Información de DB2 Connect

Nombre	Número de documento	Nombre de archivo PDF
IBM Connectivity Supplement	Sin número de documento	db2h1x81

Tabla 99. Información de DB2 Connect (continuación)

Nombre	Número de documento	Nombre de archivo PDF
IBM DB2 Connect Guía rápida de iniciación para DB2 Enterprise Edition	GC10-3774	db2c6x81
IBM DB2 Connect Quick Beginnings for DB2 Connect Personal Edition	GC09-4834	db2c1x81
IBM DB2 Connect User's Guide	SC09-4835	db2c0x81

Información de iniciación

La información de esta categoría es útil cuando se van a instalar y configurar servidores, clientes y otros productos de DB2.

Tabla 100. Información de iniciación

Nombre	Número de documento	Nombre de archivo PDF
IBM DB2 Universal Database Guía rápida de iniciación para clientes DB2	GC10-3775, no disponible en copia impresa	db2itx81
IBM DB2 Universal Database Guía rápida de iniciación para servidores DB2	GC10-3773	db2isx81
IBM DB2 Universal Database Guía rápida de iniciación para DB2 Personal Edition	GC10-3771	db2i1x81
IBM DB2 Universal Database Suplemento de instalación y configuración	GC10-3772, no disponible en copia impresa	db2iyx81
IBM DB2 Universal Database Guía rápida de iniciación para DB2 Data Links Manager	GC10-3726	db2z6x81

Información de aprendizaje

La información de aprendizaje presenta las características de DB2 y explica cómo realizar diversas tareas.

Tabla 101. Información de aprendizaje

Nombre	Número de documento	Nombre de archivo PDF
Guía de aprendizaje de Business Intelligence: Introducción al Centro de depósito de datos	Sin número de documento	db2tux81
Guía de aprendizaje de Business Intelligence: Lecciones ampliadas sobre depósito de datos	Sin número de documento	db2tax81
Information Catalog Center Tutorial	Sin número de documento	db2aix81
Guía de aprendizaje de Video Central para e-business	Sin número de documento	db2twx81
Guía de aprendizaje de Visual Explain	Sin número de documento	db2tvx81

Información sobre componentes opcionales

La información de esta categoría describe cómo trabajar con los componentes opcionales de DB2.

Tabla 102. Información sobre componentes opcionales

Nombre	Número de documento	Nombre de archivo PDF
IBM DB2 Cube Views Guía y consulta	SC10–3868	db2aax81
IBM DB2 Query Patroller Guide: Installation, Administration and Usage Guide	GC09–7658	db2dwx81
IBM DB2 Spatial Extender and Geodetic Extender Guía del usuario y de consulta	SC10-3755	db2sbx81
IBM DB2 Universal Database Data Links Manager Administration Guide and Reference	SC27-1221	db2z0x82
DB2 Net Search Extender Administración y guía del usuario **Nota:** El HTML para este documento *no* se instala desde el CD de documentación HTML.	SH10-9305	N/D

Notas del release

Las notas del release proporcionan información adicional específica del release y nivel de FixPak del producto. Las notas del release también proporcionan resúmenes de las actualizaciones de la documentación que se han incorporado en cada release, actualización y FixPak.

Tabla 103. Notas del release

Nombre	Número de documento	Nombre de archivo PDF
Notas del release de DB2	Ver nota.	Ver nota.
Notas de instalación de DB2	Sólo disponible en el CD-ROM del producto.	No disponible.

Nota: Las Notas del release están disponibles en:

- XHTML y formato de texto, en los CD de los productos
- Formato PDF, en el CD de documentación PDF

Además, las partes de las Notas del release que tratan *Problemas conocidos y soluciones alternativas* e *Incompatibilidades entre releases* también aparecen en el Centro de información de DB2.

Para ver las Notas del release en formato de texto en las plataformas basadas en UNIX, consulte el archivo `Release.Notes`. Este archivo se encuentra en el directorio `DB2DIR/Readme/%L`, donde `%L` representa el nombre de entorno nacional y `DB2DIR` representa:

- En los sistemas operativos AIX: `/usr/opt/db2_08_01`
- En los otros sistemas operativos basados en UNIX: `/opt/IBM/db2/V8.1`

Conceptos relacionados:

- "Documentación y ayuda de DB2" en la página 757

Tareas relacionadas:

- "Impresión de manuales de DB2 desde archivos PDF" en la página 775
- "Solicitud de manuales de DB2 impresos" en la página 776
- "Invocación de ayuda según contexto desde una herramienta de DB2" en la página 776

Impresión de manuales de DB2 desde archivos PDF

Puede imprimir los manuales de DB2 desde los archivos PDF del *CD de documentación PDF de DB2*. Mediante la utilización de Adobe Acrobat Reader, puede imprimir el manual entero o un rango específico de páginas.

Prerrequisitos:

Asegúrese de que tiene instalado Adobe Acrobat Reader. Si ha de instalar Adobe Acrobat Reader, está disponible desde el sitio Web de Adobe en www.adobe.com

Procedimiento:

Para imprimir un manual de DB2 desde un archivo PDF:

1. Inserte el *CD de documentación PDF de DB2*. En sistemas operativos UNIX, monte el CD de documentación PDF de DB2. Consulte el manual *Iniciación rápida* para obtener detalles sobre cómo montar un CD en sistemas operativos UNIX.
2. Abra index.htm. El archivo de abre en una ventana de navegador.
3. Pulse el título del PDF que desee ver. El PDF se abrirá en Acrobat Reader.
4. Seleccione **Archivo** → **Imprimir** para imprimir cualquier parte que desee del manual.

Conceptos relacionados:

- "Centro de información de DB2" en la página 758

Tareas relacionadas:

- "Montaje del CD-ROM (AIX)" en la publicación *Guía rápida de iniciación para servidores DB2*
- "Cómo montar el CD-ROM (HP-UX)" en la publicación *Guía rápida de iniciación para servidores DB2*
- "Montaje del CD-ROM (Linux)" en la publicación *Guía rápida de iniciación para servidores DB2*
- "Solicitud de manuales de DB2 impresos" en la página 776
- "Montaje del CD-ROM (Entorno operativo Solaris)" en la publicación *Guía rápida de iniciación para servidores DB2*

Información relacionada:

- "Documentación PDF e impresa de DB2" en la página 770

Solicitud de manuales de DB2 impresos

Si prefiere utilizar manuales en copia impresa, puede solicitarlos de tres modos distintos.

Procedimiento:

Los manuales impresos se pueden solicitar en algunos países o regiones. Compruebe, en el sitio Web de publicaciones de IBM correspondiente a su país o región, si este servicio está disponible en su país o región. Cuando las publicaciones estén disponibles para su solicitud, puede realizar lo siguiente:

- Póngase en contacto con el distribuidor autorizado o representante de marketing de IBM. Para encontrar un representante local de IBM, consulte el directorio mundial de contactos de IBM en la página Web www.ibm.com/planetwide
- Llame al teléfono `1-800-879-2755`, si está en los EE.UU. o al `1-800-IBM-4YOU`, si está en Canadá.
- Visite el Centro de publicaciones de IBM en http://www.ibm.com/shop/publications/order. La capacidad de solicitar manuales desde el Centro de publicaciones de IBM puede no estar disponible en todos los países.

En el momento en que un producto de DB2 se encuentra disponible, los manuales impresos son los mismos que aparecen en formato PDF en el *CD de documentación PDF de DB2*. El contenido de los manuales impresos que se halla en el *CD del Centro de información de DB2* también es el mismo. No obstante, existe contenido adicional en el CD del Centro de información de DB2 que no aparece en ninguno de los manuales PDF (por ejemplo, rutinas de administración de SQL y ejemplos de HTML). No todos los manuales incluidos en el CD de documentación PDF de DB2 se pueden solicitar en copia impresa.

Nota: El Centro de información de DB2 se actualiza con mayor frecuencia que los manuales PDF o en copia impresa; instale las actualizaciones de la documentación a medida que estén disponibles o consulte el Centro de información de DB2 en http://publib.boulder.ibm.com/infocenter/db2help/ para obtener la información más actualizada.

Tareas relacionadas:

- "Impresión de manuales de DB2 desde archivos PDF" en la página 775

Información relacionada:

- "Documentación PDF e impresa de DB2" en la página 770

Invocación de ayuda según contexto desde una herramienta de DB2

La ayuda según contexto proporciona información sobre las tareas o controles que están asociados con una ventana, cuaderno, asistente o asesor determinado. La ayuda según contexto está disponible desde las herramientas de administración y desarrollo de DB2 que tienen interfaces gráficas de usuario. Existen dos tipos de ayuda según contexto:

- Ayuda a la que se accede mediante el botón **Ayuda** ubicado en cada ventana o cuaderno.

- Ventanas emergentes de información, que son ventanas que se visualizan cuando el cursor del ratón se coloca sobre un campo o control o cuando se selecciona un campo o control en una ventana, cuaderno, asistente o asesor y se pulsa F1.

El botón **Ayuda** proporciona acceso a la información de visión general, de prerrequisitos y de tareas. Las ventanas emergentes de información describen los campos y controles individuales.

Procedimiento:

Para invocar la ayuda según contexto:

- Para la ayuda de ventana y de cuaderno, inicie una de las herramientas de DB2 y, luego, abra cualquier ventana o cuaderno. Pulse el botón **Ayuda** situado en la esquina inferior derecha de la ventana o del cuaderno a fin de invocar la ayuda según contexto.

  También puede acceder a la ayuda según contexto desde el elemento de menú **Ayuda** situado en la parte superior de cada uno de los centros de herramientas de DB2.

  Para los asistentes y asesores, pulse en el enlace Visión general de tareas, de la primera página, si desea ver ayuda según contexto.
- Para obtener ayuda sobre controles individuales de una ventana o un cuaderno en una ventana emergente de información, pulse el control y, a continuación, pulse **F1**. La información emergente que contiene detalles sobre el control se visualizará en una ventana amarilla.

  Nota: Para visualizar ventanas emergentes de información simplemente manteniendo el cursor del ratón sobre un campo o control, seleccione el recuadro de selección **Visualizar automáticamente ventanas emergentes de información** en la página **Documentación** del cuaderno Valores de herramientas.

  Similar a las ventanas emergentes de información, la información emergente de diagnóstico es otra forma de ayuda según contexto; en ella se incluyen reglas para la entrada de datos. La información emergente de diagnóstico se visualiza en una ventana de color morado que aparece cuando se entran datos que no son válidos o que son insuficientes. La información emergente de diagnóstico puede aparecer para:
  - Campos obligatorios.
  - Campos cuyos datos tengan un formato preciso como, por ejemplo, un campo de fecha.

Tareas relacionadas:

- “Invocación del Centro de información de DB2” en la página 767
- “Invocación de la ayuda de mensajes desde el procesador de línea de mandatos” en la página 778
- “Invocación de la ayuda de mandatos desde el procesador de línea de mandatos” en la página 778
- “Invocación de la ayuda para estados de SQL desde el procesador de línea de mandatos” en la página 779
- “Acceso al Centro de información de DB2”
- “Cómo utilizar la ayuda de DB2 UDB”
- “Establecimiento de la ubicación para acceder al Centro de información de DB2”
- “Configuración del acceso a documentación y ayuda contextual de DB2”

Invocación de la ayuda de mensajes desde el procesador de línea de mandatos

La ayuda de mensajes describe la causa de un mensaje y describe la acción que se debe realizar en respuesta al error.

Procedimiento:

Para invocar la ayuda de mensajes, abra el procesador de línea de mandatos y entre:

```
? XXXnnnnn
```

donde *XXXnnnnn* representa un identificador de mensaje válido.

Por ejemplo, `? SQL30081` muestra la ayuda acerca del mensaje SQL30081.

Conceptos relacionados:

- "Introducción a los mensajes" en la publicación *Consulta de mensajes Volumen 1*

Información relacionada:

- "db2: Mandato de invocación del procesador de línea de mandatos" en la publicación *Consulta de mandatos*

Invocación de la ayuda de mandatos desde el procesador de línea de mandatos

La ayuda de mandatos explica la sintaxis de los mandatos del procesador de línea de mandatos.

Procedimiento:

Para invocar la ayuda de mandatos, abra el procesador de línea de mandatos y entre:

```
? mandato
```

donde *mandato* representa una palabra clave o el mandato completo.

Por ejemplo, `? catalog` visualiza ayuda para todos los mandatos CATALOG, mientras que `? catalog database` visualiza ayuda solamente para el mandato CATALOG DATABASE.

Tareas relacionadas:

- "Invocación de ayuda según contexto desde una herramienta de DB2" en la página 776
- "Invocación del Centro de información de DB2" en la página 767
- "Invocación de la ayuda de mensajes desde el procesador de línea de mandatos" en la página 778
- "Invocación de la ayuda para estados de SQL desde el procesador de línea de mandatos" en la página 779

Información relacionada:

- "db2: Mandato de invocación del procesador de línea de mandatos" en la publicación *Consulta de mandatos*

Invocación de la ayuda para estados de SQL desde el procesador de línea de mandatos

DB2 Universal Database devuelve un valor de SQLSTATE para las condiciones que pueden ser el resultado de una sentencia de SQL. La ayuda de SQLSTATE explica los significados de los estados de SQL y los códigos de las clases de estados de SQL.

Procedimiento:

Para invocar la ayuda para estados de SQL, abra el procesador de línea de mandatos y entre:

```
? sqlstate o ? código de clase
```

donde *sqlstate* representa un estado de SQL válido de cinco dígitos y *código de clase* representa los dos primeros dígitos del estado de SQL.

Por ejemplo, `? 08003` visualiza la ayuda para el estado de SQL 08003, y `? 08` visualiza la ayuda para el código de clase 08.

Tareas relacionadas:

- "Invocación del Centro de información de DB2" en la página 767
- "Invocación de la ayuda de mensajes desde el procesador de línea de mandatos" en la página 778
- "Invocación de la ayuda de mandatos desde el procesador de línea de mandatos" en la página 778

Guías de aprendizaje de DB2

Las guías de aprendizaje de DB2 ayudan a conocer los diversos aspectos de DB2 Universal Database. Las guías de aprendizaje proporcionan ejercicios con instrucciones paso a paso en las áreas de desarrollo de aplicaciones, ajuste del rendimiento de las consultas de SQL, trabajo con depósitos de datos, gestión de metadatos y desarrollo de servicios Web utilizando DB2.

Antes de empezar:

Puede ver las versiones XHTML de las guías de aprendizaje desde el Centro de información en http://publib.boulder.ibm.com/infocenter/db2help/.

Algunos ejercicios de las guías de aprendizaje utilizan datos o código de ejemplo. Consulte cada guía de aprendizaje para obtener una descripción de los prerrequisitos para las tareas específicas.

Guías de aprendizaje de DB2 Universal Database:

Pulse en el título de una guía de aprendizaje de la lista siguiente para ver esa guía de aprendizaje.

Guía de aprendizaje de Business Intelligence: Introducción al Centro de depósito de datos
Realizar tareas de introducción de depósito de datos utilizando el Centro de depósito de datos.

Guía de aprendizaje de Business Intelligence: Lecciones ampliadas sobre depósito de datos
Realizar tareas avanzadas de depósito de datos utilizando el Centro de depósito de datos.

Information Catalog Center Tutorial
Crear y gestionar un catálogo de información para localizar y usar metadatos utilizando el Centro de catálogos de información.

Guía de aprendizaje de Visual Explain
Analizar, optimizar y ajustar sentencias de SQL para obtener un mejor rendimiento al utilizar Visual Explain.

Información de resolución de problemas de DB2

Existe una gran variedad de información para la resolución de problemas y la determinación de problemas para ayudarle a utilizar los productos DB2®.

Documentación de DB2
La información de resolución de problemas se puede encontrar en todo el Centro de información de DB2, así como en todos los manuales PDF que componen la biblioteca de DB2. Puede consultar la rama sobre soporte y resolución de problemas, del árbol de navegación del Centro de información de DB2 (en el panel izquierdo de la ventana del navegador), para obtener un listado completo de la documentación de resolución de problemas de DB2.

Sitio Web de soporte técnico de DB2
Consulte el sitio Web de soporte técnico de DB2 si tiene problemas y desea obtener ayuda para encontrar las causas y las soluciones posibles. El sitio de soporte técnico tiene enlaces con las últimas publicaciones de DB2, notas técnicas, Informes autorizados de análisis del programa (APAR), FixPaks y el listado más reciente de códigos de error internos de DB2, además de otros recursos. Puede buscar en esta base de conocimiento para encontrar posibles soluciones a los problemas.

Para acceder al sitio Web de soporte de DB2, vaya a http://www.ibm.com/software/data/db2/udb/winos2unix/support

DB2 Problem Determination Tutorial Series (Serie de guías de aprendizaje para la determinación de problemas de DB2)
Consulte el sitio Web DB2 Problem Determination Tutorial Series para encontrar información sobre cómo identificar y resolver rápidamente los problemas que puedan surgir mientras trabaje con DB2. Una de las guías de aprendizaje ofrece una presentación de los recursos y las herramientas de determinación de problemas de DB2 disponibles y le ayuda a decidir cuándo utilizarlos. Otras de las guías de aprendizaje tratan temas relacionados como, por ejemplo, ″Determinación de problemas del motor de base de datos″, ″Determinación de problemas de rendimiento″ y ″Determinación de problemas de aplicaciones″.

Consulte el conjunto completo de guías de aprendizaje de determinación de problemas de DB2 en el sitio de soporte técnico de DB2 de http://www.ibm.com/software/data/support/pdm/db2tutorials.html

Conceptos relacionados:

- "Centro de información de DB2" en la página 758
- "Introduction to problem determination - DB2 Technical Support tutorial" en la *Guía de resolución de problemas*

Accesibilidad

Las características de accesibilidad ayudan a los usuarios con discapacidades físicas, por ejemplo movilidad o visión limitada, a utilizar los productos de software satisfactoriamente. La lista siguiente especifica las características de accesibilidad principales de los productos de DB2® Versión 8:

- Toda la funcionalidad de DB2 está disponible utilizando el teclado para la navegación en lugar del ratón. Si desea más información, consulte el apartado "Entrada de teclado y navegación".
- Puede personalizar el tamaño y color de los fonts en las interfaces de usuario de DB2. Si desea más información, consulte el apartado "Pantalla accesible".
- Los productos de DB2 dan soporte a aplicaciones de accesibilidad que utilizan la API de accesibilidad de Java™. Si desea más información, consulte el apartado "Compatibilidad con tecnologías de asistencia" en la página 782.
- La documentación de DB2 se proporciona en un formato accesible. Si desea más información, consulte el apartado "Documentación accesible" en la página 782.

Entrada de teclado y navegación

Entrada de teclado

Puede trabajar con las herramientas de DB2 utilizando solamente el teclado. Puede utilizar teclas o combinaciones de teclas para llevar a cabo operaciones que también se pueden realizar con el ratón. Las pulsaciones estándares del sistema operativo se utilizan para operaciones estándares del sistema operativo.

Para obtener más información sobre el uso de teclas o combinaciones de teclas al realizar operaciones, consulte Accesos directos y aceleradores del teclado.

Navegación de teclado

Puede navegar por la interfaz de usuario de las herramientas de DB2 mediante teclas o combinaciones de teclas.

Para obtener más información sobre el uso de teclas o combinaciones de teclas al navegar por las herramientas de DB2, consulte Accesos directos y aceleradores del teclado.

Foco del teclado

En los sistemas operativos UNIX®, se resalta el área de la ventana activa en la que las pulsaciones tendrán efecto.

Pantalla accesible

Las herramientas de DB2 presentan características que mejoran la accesibilidad de los usuarios con poca visión u otras discapacidades visuales. Estas mejoras de la accesibilidad incluyen soporte para propiedades de font personalizables.

Valores de font

Puede seleccionar el color, tamaño y font del texto en menús y ventanas de diálogo utilizando el cuaderno Valores de herramientas.

Para obtener más información sobre cómo especificar valores de font, consulte Modificación de fonts para menús y texto.

No dependencia del color

No es necesario distinguir los colores para utilizar cualquiera de las funciones de este producto.

Compatibilidad con tecnologías de asistencia

Las interfaces de las herramientas de DB2 dan soporte a la API de accesibilidad de Java, que le permite utilizar lectores de pantalla y otras tecnologías de asistencia con los productos de DB2.

Documentación accesible

La documentación de DB2 se proporciona en formato XHTML 1.0, que se puede visualizar en la mayoría de los navegadores Web. XHTML le permite visualizar la documentación de acuerdo con las preferencias de pantalla establecidas en el navegador. También permite utilizar lectores de pantalla y otras tecnologías de asistencia.

Los diagramas de sintaxis se proporcionan en formato decimal con puntos. Este formato sólo está disponible si se accede a la documentación en línea mediante un lector de pantalla.

Conceptos relacionados:

- "Diagramas de sintaxis en formato decimal con puntos" en la página 782

Tareas relacionadas:

- "Accesos directos y aceleradores del teclado"
- "Modificación de fonts para menús y texto"

Diagramas de sintaxis en formato decimal con puntos

Se proporcionan diagramas de sintaxis en formato decimal con puntos para los usuarios que acceden al Centro de información utilizando un lector de pantalla.

En formato decimal con puntos, cada elemento de sintaxis se escribe en una línea distinta. Si dos o más elementos de sintaxis siempre aparecen juntos (o siempre están ausentes los dos a la vez), pueden aparecer en la misma línea, puesto que se pueden considerar un elemento de sintaxis compuesto.

Cada línea empieza por un número decimal con puntos; por ejemplo, 3 ó 3.1 ó 3.1.1. Para oír estos números correctamente, asegúrese de que su lector de pantalla esté configurado para leer la puntuación. Todos los elementos de sintaxis que tienen el mismo número decimal con puntos (por ejemplo, todos los elementos de sintaxis que tienen el número 3.1) son alternativas mutuamente excluyentes. Si oye las líneas 3.1 USERID y 3.1 SYSTEMID, sabrá que la sintaxis puede incluir o USERID o SYSTEMID, pero no ambos.

El nivel de numeración decimal con puntos denota el nivel jerárquico. Por ejemplo, si un elemento de sintaxis con el número decimal con puntos 3 va seguido de una serie de elementos de sintaxis con el número decimal 3.1, todos los elementos de sintaxis con la numeración 3.1 son subordinados de los elementos de sintaxis identificados por el número 3.

Junto a los números decimales con puntos se utilizan determinados símbolos y palabras para añadir información sobre los elementos de sintaxis. A veces, estos símbolos y palabras pueden aparecer al principio del propio elemento. Para facilitar la identificación, si la palabra o el símbolo forman parte del elemento de sintaxis, van precedidos por una barra inclinada invertida (\). El símbolo * se puede utilizar junto a un número decimal con puntos para indicar que el elemento de sintaxis se repite. Por ejemplo, el elemento de sintaxis *FILE con el número decimal con puntos 3 adopta el formato 3 * FILE. El formato 3* FILE indica que el elemento de sintaxis FILE se repite. El formato 3* * FILE indica que el elemento de sintaxis * FILE se repite.

Los caracteres como las comas, que se utilizan para separar una serie de elementos de sintaxis, se muestran en la sintaxis justo antes de los elementos que separan. Estos caracteres pueden aparecer en la misma línea que cada elemento o en una línea distinta con el mismo número decimal con puntos que los elementos en cuestión. En la línea también puede aparecer otro símbolo que proporcione información sobre los elementos de sintaxis. Por ejemplo, las líneas 5.1*, 5.1 LASTRUN y 5.1 DELETE significan que si se utiliza más de uno de los elementos de sintaxis LASTRUN y DELETE, los elementos deben estar separados por comas. Si no hay ningún separador, suponga que utiliza un espacio en blanco para separar cada elemento de sintaxis.

Si un elemento de sintaxis va precedido del símbolo %, esto indica una referencia que está definida en cualquier otro lugar. La serie que aparece después del símbolo % es el nombre de un fragmento de sintaxis en lugar de un literal. Por ejemplo, la línea 2.1 %OP1 significa que se debe hacer referencia al fragmento de sintaxis separado OP1.

Junto a los números decimales con puntos se utilizan los símbolos y las palabras siguientes:

- ? indica un elemento de sintaxis opcional. Un número decimal con puntos seguido del símbolo ? indica que todos los elementos de sintaxis con un número decimal con puntos correspondiente y elementos de sintaxis subordinados son opcionales. Si sólo hay un elemento de sintaxis con un número decimal con puntos, el símbolo ? aparecerá en la misma línea que el elemento de sintaxis (por ejemplo, 5? NOTIFY). Si hay más de un elemento de sintaxis con un número decimal con puntos, el símbolo ? aparecerá en una línea propia, seguido de los elementos de sintaxis opcionales. Por ejemplo, si oye las líneas 5 ?, 5 NOTIFY y 5 UPDATE, sabrá que los elementos de sintaxis NOTIFY y UPDATE son opcionales; es decir, puede seleccionar uno o ninguno de dichos elementos. El símbolo ? es equivalente a una línea de desvío de un diagrama de vías.
- ! indica un elemento de sintaxis por omisión. Un número decimal con puntos seguido del símbolo ! y un elemento de sintaxis indica que el elemento de sintaxis es la opción por omisión para todos los elementos de sintaxis que comparten el mismo número decimal con puntos. Sólo uno de los elementos de sintaxis que comparten el mismo número decimal con puntos puede especificar un símbolo !. Por ejemplo, si oye las líneas 2? FILE, 2.1! (KEEP) y 2.1 (DELETE), sabrá que (KEEP) es la opción por omisión correspondiente a la palabra clave FILE. En este ejemplo, si incluye la palabra clave FILE pero no especifica ninguna opción, se aplicará la opción por omisión KEEP. También se aplicará una opción por omisión al siguiente número decimal con puntos más alto. En este ejemplo, si se omite la palabra clave FILE, se utiliza el valor por omisión FILE(KEEP). No obstante, si oye las líneas 2? FILE, 2.1, 2.1.1! (KEEP) y 2.1.1 (DELETE), la opción por omisión KEEP sólo se aplicará al siguiente número

decimal con puntos más alto, 2.1 (que no tiene una palabra clave asociada) y no se aplicará a 2? FILE. Si se omite la palabra clave FILE, no se utilizará nada.

- * indica un elemento de sintaxis que se puede repetir 0 o más veces. Un número decimal con puntos seguido del símbolo * indica que este elemento de sintaxis se puede utilizar cero o más veces; es decir, es opcional y se puede repetir. Por ejemplo, si oye la línea 5.1* data area, sabrá que puede incluir un área de datos, más de un área de datos o ningún área de datos. Si oye las líneas 3*, 3 HOST y 3 STATE, sabrá que puede incluir HOST, STATE, los dos juntos o ninguno de los dos.

  Notas:

  1. Si un número decimal con puntos tiene un asterisco (*) al lado y sólo hay un elemento con dicho número decimal con puntos, podrá repetir el mismo elemento más de una vez.
  2. Si un número decimal con puntos tiene un asterisco al lado y hay varios elementos que tienen dicho número decimal con puntos, podrá utilizar más de un elemento de la lista, pero no podrá utilizar los elementos más de una vez cada uno. En el ejemplo anterior, podría escribir HOST STATE pero no podría escribir HOST HOST.
  3. El símbolo * es equivalente a una línea de bucle de retorno de un diagrama de sintaxis de vías.

- + indica un elemento de sintaxis que se debe incluir una o más veces. Un número decimal con puntos seguido del símbolo + indica que este elemento de sintaxis se debe incluir una o más veces; es decir, se debe incluir como mínimo una vez y se puede repetir. Por ejemplo, si oye la línea 6.1+ data area, deberá incluir como mínimo un área de datos. Si oye las líneas 2+, 2 HOST y 2 STATE, sabrá que debe incluir HOST, STATE o ambos. De manera similar al símbolo *, el símbolo + sólo puede repetir un elemento determinado si éste es el único elemento que tiene el número decimal con puntos en cuestión. El símbolo +, al igual que el símbolo *, es equivalente a una línea de bucle de retorno de un diagrama de sintaxis de vías.

Conceptos relacionados:

- "Accesibilidad" en la página 781

Tareas relacionadas:

- "Accesos directos y aceleradores del teclado"

Información relacionada:

- "Cómo se leen los diagramas de sintaxis" en la publicación *Consulta de SQL, Volumen 2*

Certificación Common Criteria de productos DB2 Universal Database

Se está evaluando DB2 Universal Database para obtener la certificación Common Criteria en el nivel de garantía de evaluación 4 (EAL4). Para más información acerca de Common Criteria, consulte el sitio Web de Common Criteria en: http://niap.nist.gov/cc-scheme/.

Apéndice F. Avisos

Es posible que IBM no comercialice en todos los países algunos productos, servicios o características descritos en este manual. Consulte al representante local de IBM para obtener información sobre los productos y servicios que actualmente pueden adquirirse en su zona. Cualquier referencia a un producto, programa o servicio de IBM no pretende afirmar ni implicar que sólo se pueda utilizar dicho producto, programa o servicio de IBM. En su lugar se puede utilizar cualquier producto, programa o servicio funcionalmente equivalente que no vulnere ninguno de los derechos de propiedad intelectual de IBM. Sin embargo, es responsabilidad del usuario evaluar y verificar el funcionamiento de cualquier producto, programa o servicio que no sea de IBM.

IBM puede tener patentes o solicitudes de patentes en tramitación que afecten al tema tratado en este documento. La posesión de este documento no confiere ninguna licencia sobre dichas patentes. Puede realizar consultas sobre licencias escribiendo a:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
EE.UU.

Para realizar consultas sobre licencias referentes a información de doble byte (DBCS), puede ponerse en contacto con el Departamento de Propiedad Intelectual de IBM de su país/región o escribir a:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokio 106, Japón

El párrafo siguiente no es aplicable al Reino Unido ni a ningún país/región en donde tales disposiciones sean incompatibles con la legislación local:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROPORCIONA ESTA PUBLICACIÓN "TAL CUAL", SIN GARANTÍA DE NINGUNA CLASE, NI EXPLÍCITA NI IMPLÍCITA, INCLUIDAS, PERO SIN LIMITARSE A ELLAS, LAS GARANTÍAS IMPLÍCITAS DE NO VULNERACIÓN DE DERECHOS, COMERCIALIZACIÓN O IDONEIDAD PARA UN FIN DETERMINADO. Algunos estados no permiten la exclusión de garantías expresas o implícitas en determinadas transacciones, por lo que es posible que esta declaración no sea aplicable en su caso.

Esta publicación puede contener inexactitudes técnicas o errores tipográficos. Periódicamente se efectúan cambios en la información aquí contenida; dichos cambios se incorporarán a las nuevas ediciones de la publicación. IBM puede efectuar, en cualquier momento y sin previo aviso, mejoras y cambios en los productos y programas descritos en esta publicación.

Las referencias hechas en esta publicación a sitios Web que no son de IBM se proporcionan sólo para la comodidad del usuario y no constituyen un aval de esos

sitios Web. La información contenida en esos sitios Web no forma parte de la información del presente producto IBM y el usuario es responsable de la utilización de dichos sitios Web.

IBM puede utilizar o distribuir cualquier información que se le facilite de la manera que considere adecuada, sin contraer por ello ninguna obligación con el remitente.

Los licenciatarios de este programa que deseen obtener información sobre él con el fin de habilitar: (i) el intercambio de información entre programas creados de forma independiente y otros programas (incluido éste) y (ii) el uso mutuo de la información intercambiada, deben ponerse en contacto con:

IBM Canada Limited
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
CANADÁ

Dicha información puede estar disponible, sujeta a los términos y condiciones apropiados, incluido en algunos casos el pago de una tarifa.

El programa bajo licencia descrito en este documento y todo el material bajo licencia asociado a él, los proporciona IBM según los términos del Acuerdo de Cliente de IBM, el Acuerdo Internacional de Programas Bajo Licencia de IBM o cualquier acuerdo equivalente entre el usuario e IBM.

Los datos de rendimiento contenidos en este documento se obtuvieron en un entorno controlado. Por lo tanto, los resultados obtenidos en otros entornos operativos pueden variar significativamente. Algunas mediciones pueden haberse realizado en sistemas experimentales y no es seguro que estas mediciones sean las mismas en los sistemas disponibles comercialmente. Además, algunas mediciones pueden haberse calculado mediante extrapolación. Los resultados reales pueden variar. Los usuarios del presente manual deben verificar los datos aplicables para su entorno específico.

La información referente a productos que no son de IBM se ha obtenido de los proveedores de esos productos, de sus anuncios publicados o de otras fuentes disponibles públicamente. IBM no ha probado esos productos y no puede confirmar la exactitud del rendimiento, la compatibilidad ni ninguna otra afirmación referente a productos que no son de IBM. Las preguntas sobre las prestaciones de productos que no son de IBM deben dirigirse a los proveedores de esos productos.

Todas las declaraciones de intenciones de IBM están sujetas a cambio o cancelación sin previo aviso, y sólo representan objetivos.

Este manual puede contener ejemplos de datos e informes que se utilizan en operaciones comerciales diarias. Para ilustrarlos de la forma más completa posible, los ejemplos incluyen nombres de personas, empresas, marcas y productos. Todos estos nombres son ficticios y cualquier similitud con nombres y direcciones utilizados por una empresa real es totalmente fortuita.

LICENCIA DE COPYRIGHT:

Este manual puede contener programas de aplicaciones de ejemplo escritos en lenguaje fuente, que muestran técnicas de programación en diversas plataformas operativas. Puede copiar, modificar y distribuir estos programas de ejemplo como desee, sin pago alguno a IBM, con la intención de desarrollar, utilizar, comercializar o distribuir programas de aplicaciones de acuerdo con la interfaz de programación de aplicaciones correspondiente a la plataforma operativa para la que están escritos los programas de ejemplo. Estos ejemplos no se han probado exhaustivamente bajo todas las condiciones. Por lo tanto, IBM no puede asegurar ni implicar la fiabilidad, utilidad o función de estos programas.

Cada copia o parte de estos programas de ejemplo o cualquier trabajo derivado debe incluir una nota de copyright como la siguiente:

© (*nombre de la empresa*) (*año*). Partes de este código proceden de programas de ejemplo de IBM Corp. © Copyright IBM Corp. *_entre el o los años_*. Reservados todos los derechos.

Marcas registradas

Los términos siguientes son marcas registradas de International Business Machines Corporation en los EE.UU. y/o en otros países y se han utilizado como mínimo en uno de los documentos de la biblioteca de documentación de DB2 UDB.

ACF/VTAM
AISPO
AIX
AIXwindows
AnyNet
APPN
AS/400
BookManager
C Set++
C/370
CICS
Database 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Extenders
DB2 OLAP Server
DB2 Information Integrator
DB2 Query Patroller
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
eServer
Extended Services
FFST
First Failure Support Technology
IBM
IMS
IMS/ESA
iSeriesLAN Distance
MVS
MVS/ESA
MVS/XA
Net.Data
NetView
OS/390
OS/400
PowerPC
pSeries
QBIC
QMF
RACF
RISC System/6000
RS/6000
S/370
SP
SQL/400
SQL/DS
System/370
System/390
SystemView
Tivoli
VisualAge
VM/ESA
VSE/ESA
VTAM
WebExplorer
WebSphere
WIN-OS/2
z/OS
zSeries

Los términos siguientes son marcas registradas de otras empresas y se han utilizado como mínimo en uno de los documentos de la biblioteca de documentación de DB2 UDB:

Microsoft, Windows, Windows NT y el logotipo de Windows son marcas registradas de Microsoft Corporation en los EE.UU. y/o en otros países.

Intel y Pentium son marcas registradas de Intel Corporation en los EE.UU. y/o en otros países.

Java y todas las marcas registradas basadas en Java son marcas registradas de Sun Microsystems, Inc. en los EE.UU. y/o en otros países.

UNIX es marca registrada de The Open Group en los EE.UU. y/o en otros países.

Otros nombres de empresas, productos o servicios, pueden ser marcas registradas o marcas de servicio de otras empresas.

Índice

Caracteres Especiales

#ifdefs
  restricciones para C/C++ 165

A

accesibilidad
  características 781
  diagramas de sintaxis con puntos decimales 782
acceso a paquetes Java
  JDBC 290
  SQLJ 346
ACQUIRE, sentencia
  no soportada en DB2 UDB 750
activadores
  actualizaciones anteriores 48
  actualizaciones posteriores 48
  consideración sobre lógica de aplicación 49
  control de relaciones de datos 47
  visión general 24
actualización
  Centro de información de DB2 768
actualizaciones
  en tablas DB2, JDBC 303
actualizaciones múltiples
  finalidad 672
  parámetros de configuración 676
  precompilación de aplicaciones 674
  sentencias de SQL en aplicaciones de actualización múltiple 673
  soporte de DB2 Connect 749
  visión general 672
actualizaciones por lotes
  aplicación JDBC 330
  aplicación SQLJ 383
agrupación de conexiones
  ADO 695
  ODBC 695
almacenamiento
  asignación para páginas de códigos desiguales 660
  asociación para albergar filas 129
  declarar suficientes entidades SQLVAR 125
almacenamientos intermedios, tamaño
  para inserción en almacenamiento intermedio 699
API
  comparación de las implementaciones de JDBC 407
  de plug-in 599, 608
  para plug-in de seguridad 597, 601, 602, 606, 607, 608, 615, 616, 617, 618, 620, 622, 624, 625, 627, 629, 631
API Bind
  creación de paquetes 71
API Bind *(continuación)*
  vinculación diferida 78
API de transacciones de Java (JTA) 514
API GET ERROR MESSAGE
  recuperación de mensajes de error 112
  variables REXX predefinidas 534
aplicación JDBC
  actualizaciones por lotes 330
  conectar con fuente de datos 291
  crear y modificar objetos DB2 301
  declarar variables 290
  definir nivel de aislamiento para 299
  ejecutar SQL 300
  ejemplo 287
  pasos básicos 287
  recuperar datos de columnas de identidad 321
  uso de puntos de rescate 320
aplicación SQLJ
  actualizaciones por lotes 383
  comentarios 348
  control de la ejecución de sentencias 381
  crear y modificar objetos DB2 357
  declarar variables 347
  definir nivel de aislamiento para 354
  DELETE de posición 363
  ejecutar SQL 356
  ejemplo 343
  iterador de nombre, usar 358
  manejo de errores 371
  pasos básicos 343
  recuperar datos de tablas DB2 357
  UPDATE de posición 363
  uso de puntos de rescate 354
  uso de un iterador desplazable 389
aplicaciones
  actualización múltiple, precompilación 674
  ADO
    cursores desplazables actualizables 257
    limitaciones 257
  bucle 712
  conectar con fuentes de datos, IBM OLE DB Provider 262
  en entornos de sistema principal iSeries 739
  funciones de MQSeries 17
  funciones de programación de DB2 18
  gestión de transacciones con puntos de rescate 680
  herramientas de DB2 para desarrollar 3
  herramientas para crear aplicaciones Web 16
  interfaces de programación soportadas 5
  Interfaz XA de X/Open, enlace 691
aplicaciones *(continuación)*
  puntos de rescate, restricciones 685
  soportado en Java 2 Enterprise Edition 511
  soportado por IBM OLE DB Provider 242
  suspendida 712
  Visual Basic, conectar con fuente de datos 257
aplicaciones ADO
  agrupación de conexiones, con MTS y COM+ 695
  cursores desplazables actualizables 257
  limitaciones 257
  palabras clave de serie de conexión 256
  procedimientos almacenados 257
  soporte de IBM OLE DB Provider para métodos y propiedades ADO 258
aplicaciones C/C++
  acceso a bases de datos de varias hebras 187
  compilar y enlazar, IBM OLE DB Provider 261
  conexión con fuentes de datos, IBM OLE DB Provider 262
aplicaciones compiladas, creación de paquetes 65
aplicaciones REXX 541
aplicaciones Web
  herramientas para crear 16
APPC (Comunicación Avanzada Programa a Programa)
  manejo de interrupciones 111
archivos
  declaraciones de referencia en C/C++ 164
archivos de entrada para C/C 146
archivos de entrada y salida
  COBOL 194
  FORTRAN 216
archivos de mensajes
  definición 67
archivos de salida
  C/C++ 146
archivos de vinculación
  compatibilidad con versiones anteriores 77
  opciones de precompilación 67
  REXX 542
  soporte para aplicaciones REXX 542
archivos fuente
  crear 63
archivos include
  requisitos
    C/C 147
    COBOL 194
    FORTRAN 216

archivos include *(continuación)*
SQL
para C/C 147
para COBOL 194
para FORTRAN 216
SQL1252A
COBOL 194
FORTRAN 216
SQL1252B
COBOL 194
FORTRAN 216
SQLADEF para C/C++ 147
SQLAPREP
para C/C 147
para COBOL 194
para FORTRAN 216
SQLCA
para C/C 147
para COBOL 194
para FORTRAN 216
SQLCA_92
COBOL 194
FORTRAN 216
SQLCACN
FORTRAN 216
SQLCACS
FORTRAN 216
SQLCLI para C/C++ 147
SQLCLI1 para C/C++ 147
SQLCODES
para C/C 147
para COBOL 194
para FORTRAN 216
SQLDA
COBOL 194
para C/C 147
para FORTRAN 216
SQLDACT
FORTRAN 216
SQLE819A
para C/C 147
para COBOL 194
para FORTRAN 216
SQLE819B
para C/C 147
para COBOL 194
para FORTRAN 216
SQLE850A
para C/C 147
para COBOL 194
para FORTRAN 216
SQLE850B
para C/C 147
para COBOL 194
para FORTRAN 216
SQLE932A
para C/C 147
para COBOL 194
para FORTRAN 216
SQLE932B
para C/C 147
para COBOL 194
para FORTRAN 216
SQLEAU
para C/C 147
para COBOL 194
para FORTRAN 216
archivos include *(continuación)*
SQLENV
COBOL 194
FORTRAN 216
para C/C 147
SQLETSD
COBOL 194
SQLEXT para C/C++ 147
SQLJACB para C/C++ 147
SQLMON
COBOL 194
FORTRAN 216
para C/C 147
SQLMONCT
para COBOL 194
SQLSTATE
para C/C 147
para COBOL 194
para FORTRAN 216
SQLSYSTM para C/C++ 147
SQLUDF para C/C++ 147
SQLUTBCQ
COBOL 194
SQLUTBSQ
COBOL 194
SQLUTIL
para C/C 147
para COBOL 194
para FORTRAN 216
SQLUV para C/C++ 147
SQLUVEND para C/C++ 147
SQLXA para C/C++ 147
ubicación
en C/C 149
en COBOL 196
en FORTRAN 219
área de comunicaciones de SQL (SQLCA) 34
ARI en campo SQLERRP
DB2 para VSE VM 742
ASCII
datos de bytes mixtos 741
orden de clasificación 745
atajos de teclado
soporte para 781
ATOMIC, SQL compuesto
soporte de DB2 Connect 748
autenticación
plug-ins
API para comprobar si existe ID de autenticación 631
API para inicializar un plug-in de autenticación de cliente 615
API para liberar recursos de autenticación de cliente 616
API para validar contraseñas 622
autenticación de ID de usuario/contraseña 608
para inicializar un plug-in de autenticación de cliente 615
ubicaciones de bibliotecas 573
por plug-in de seguridad 569
aviso de SQL
manejo en JDBC 312, 313
manejo en SQLJ 372
ayuda
para mandatos 778
ayuda *(continuación)*
para mensajes 778
para sentencias de SQL 779
visualizar 767, 769
ayuda para mandatos
invocar 778
ayuda para mensajes
invocar 778
ayuda para sentencias de SQL
invocar 779

B

base de datos
conectar aplicación con 36
bases de datos
acceso
varias hebras 187
conectar con Perl 525
crear
orden de clasificación 642
utilizando contextos distintos 187
BatchUpdateException
recuperar información de, JDBC 332
bibliotecas
bibliotecas de plug-in de seguridad 587
restricciones para 588
BIGINT, tipo de datos de SQL
C/C++, conversión 180
COBOL 210
BINARY, tipos de datos de COBOL 213
blob, tipo de C/C++ 180
BLOB, tipo de datos
C/C++, conversión 180
COBOL 210
REXX 539
blob_file, tipo de C/C++ 180
BLOB-FILE, tipo de COBOL 210
blob_locator, tipo de C/C++ 180
BLOB-LOCATOR, tipo de COBOL 210
Bloque de descriptor de base de datos (SQLEDBDESC), especificación de órdenes de clasificación 642
bloqueo
error de inserción en almacenamiento intermedio 702
bloqueo a nivel de fila
entornos de sistema principal e iSeries 746
bloqueo a nivel de página
entornos de sistema principal e iSeries 746
bloqueos
liberación de cursor 98
nivel de fila 746
nivel de página 746
tiempo de espera 746
buscar
documentación de DB2 758

C

C
series terminadas en nulo 744

calificación y operaciones de miembros en C/C++ 173
CALL, sentencias
  CALL USING DESCRIPTOR 747
  plataformas soportadas 747
campo SQLSTATE 110
campos largos
  diferencias entre plataformas 741
  inserciones en almacenamiento intermedio, restricción 704
caracteres
  sustitución durante conversión de página de códigos 651
caracteres de desplazamiento a teclado ideográfico, diferencias entre plataformas 741
catálogo SYSIBM.SYSROUTINES (VM/VSE) 747
catálogos del sistema
  entornos de sistema principal e iSeries 747
Centro de desarrollo
  características 20
  visión general 20
Centro de información
  instalación 760, 762, 765
Centro de información de DB2 758
  actualización 768
  invocar 767
  ver en idiomas diferentes 769
cerrar conexión
  fuente de datos JDBC 300
  fuente de datos SQLJ 356
cerrar inserción colocada en almacenamiento intermedio 699
CICS (Customer Information Control System)
  diferencias de aplicaciones según la plataforma 739
CICS SYNCPOINT ROLLBACK, mandato 688
clasificación
  juegos de códigos en chino (tradicional) 659
  juegos de códigos en japonés 659
  orden de clasificación 642, 745
  orden de resultados 745
cláusula de asignación, SQLJ 437
cláusula de contexto, SQLJ 434
cláusula de conversión a iterador, SQLJ 438
cláusula de declaración de conexión, SQLJ 430, 431
cláusula de sentencia, SQLJ 434
cláusula ejecutable, SQLJ 433
cláusula FOR UPDATE 102
cláusula implements, SQLJ 428
cláusula ORDER BY
  orden de clasificación 745
cláusula SET-TRANSACTION, SQLJ 436
cláusula SQLJ 427
cláusula with, SQLJ 429
claves
  foránea
    diferencias entre plataformas 745
  primaria 745
claves foráneas
  diferencias entre plataformas 745
claves primarias
  diferencias entre plataformas 745
CLI (interfaz de nivel de llamada)
  archivos de rastreo 500
  frente a SQL dinámico incorporado 140
  recurso de rastreo 494
CLI/ODBC/JDBC
  rastreo
    archivos 500
    recurso 494
clientes
  invocar procedimientos almacenados en REXX 545
CLOB (character large object)
  C/C++, conversión 180
  tipo de datos
    C/C++ 184
    COBOL 210
    FORTRAN 227
    REXX 539
    variables indicadoras 93
CLOB-FILE, tipo de COBOL 210
clob_file, tipo de datos de C/C++ 180
CLOB-LOCATOR, tipo de COBOL 210
clob_locator, tipo de datos de C/C++ 180
COBOL, lenguaje
  archivos de entrada y salida 194
  archivos include 194
  consideraciones sobre EUC en chino (tradicional) 213
  consideraciones sobre EUC en japonés 213
  consideraciones sobre programación 193
  declaración de variables del lenguaje principal 200
  declaraciones de datos LOB 204
  declaraciones de localizadores de LOB 205
  declaraciones de referencias de archivos 206
  declarar variables del lenguaje principal de gráficos 203
  estructuras de sistema principal 206
  FOR BIT DATA 213
  no hay soporte para acceso a bases de datos de varias hebras 193
  nomenclatura de variables del lenguaje principal 199
  normas para las variables indicadoras 204
  REDEFINES 209
  referencia a variables del lenguaje principal 198
  restricciones 193
  restricciones orientadas a objetos 214
  sentencias de SQL incorporado 61, 196
  tablas de indicadores 208
  tipos de datos 210
  variables de lenguaje principal de tipo carácter y longitud fija, sintaxis 201
COBOL, lenguaje *(continuación)*
  variables del lenguaje principal numéricas 200
  variables SQLCODE 213
  variables SQLSTATE 213
COBOL, tipos de datos
  BINARY 213
  BLOB 210
  BLOB-FILE 210
  BLOB-LOCATOR 210
  CLOB 210
  CLOB-FILE 210
  CLOB-LOCATOR 210
  COMP 213
  COMP-1 210
  COMP-3 210
  COMP-4 213
  COMP-5 210
  DBCLOB 210
  DBCLOB-FILE 210
  DBCLOB-LOCATOR 210
  PICTURE (PIC), cláusula 210
  USAGE cláusula 210
códigos de error, JDBC
  para errores del controlador JDBC universal de DB2 468
códigos de finalización 34
códigos de resultados 34
códigos de retorno
  declaración del SQLCA 34
  estructura de SQLCA 110
códigos de territorio
  SQLERRMC, campo de SQLCA 742
códigos satisfactorios 34
coherencia
  de datos 37
colocación en serie
  estructuras de datos 188
  SQL, ejecución de sentencias 187
columna, tipos de
  crear
    C/C++ 180
    COBOL 210
    FORTRAN 227
columna LOB
  elegir tipos de datos Java compatibles, JDBC 316
  elegir tipos de datos Java compatibles, SQLJ 376
columnas
  derivadas 715
  establecimiento de valores nulos 91
  generadas 715
  identidad 716
  incluir columnas 721
  tipos de datos de SQL soportados 93
  utilización de variables indicadoras en columnas de datos que pueden tener valores null 95
columnas de identidad
  comparación con objetos en secuencia 726
  finalidad 716
  recuperar datos de, JDBC 321
columnas derivadas
  finalidad 715

columnas generadas
finalidad 715
COM+
gestor de transacciones 691
proceso de transacciones 693
reutilizar conexión 693
soporte para acoplamiento débil 693
tiempo de espera de transacción 694
com.ibm.db2.jcc.DB2BaseDataSource
métodos 446
propiedades 446
com.ibm.db2.jcc.DB2DatabaseMetaData
métodos 446
com.ibm.db2.jcc.DB2Diagnosable
métodos 446
com.ibm.db2.jcc.DB2Driver
métodos 446
com.ibm.db2.jcc.DB2ExceptionFormatter
métodos 446
com.ibm.db2.jcc.DB2JccDataSource
métodos 446
com.ibm.db2.jcc.DB2SimpleDataSource
métodos 446
propiedades 446
com.ibm.db2.jcc.DB2Sqlca
métodos 446
comentarios
aplicación SQLJ 348
sentencia de SQL incorporado 531
SQL, normas 150, 196, 219
commit
transacción JDBC 300
transacción SQLJ 354
COMMIT WORK RELEASE, sentencia no soportada en DB2 Connect 750
Common Language Runtime
rutinas
tipos de datos de SQL soportados en 238
comparación de caracteres 639
compilación
visión general 70
comportamiento de definición, DYNAMICRULES 122
comportamiento de ejecución, DYNAMICRULES 122
comportamiento de invocación, DYNAMICRULES 122
comportamiento de vinculación, DYNAMICRULES 122
conectar
con fuente de datos utilizando DataSource 297
con fuente de datos utilizando DriverManager
controlador JDBC de DB2 de tipo 2 292
controlador JDBC universal de DB2 294
con fuente de datos utilizando SQLJ 349
conexiones
agrupación
WebSphere 563, 564
CONNECT nula 742
implícita
diferencias entre plataformas 742
conexiones *(continuación)*
sentencia CONNECT RESET 742
sentencia CONNECT TO 742
uso en JDBC 299
conexiones implícitas
diferencias entre plataformas 742
confirmación de cambios
tablas 38
confirmación en dos fases
actualización
varias bases de datos 672
conjunto de filas de esquema
IBM OLE DB Provider 243
conjunto de resultados actualizable
JDBC 335
conjunto de resultados con capacidad de retención, JDBC 335
conjunto de resultados desplazable
JDBC 335
restricción para tipos de datos 335
consideraciones sobre programación
acceso a servidores de sistema principal, AS/400 o iSeries 677
C/C++ 145
COBOL 193
entornos 27
FORTRAN 215
infraestructura de seudocódigo 41
interfaces soportadas 5
interfaz XA de X/Open 688
REXX 529
tipos de variables, control de valores de datos 46
constantes gráficas
juegos de códigos en chino (tradicional) 659
juegos de códigos en japonés 659
consultas
actualizable 102
suprimible 102
contenedores
Java 2 Enterprise Edition 512
contexto de ejecución
SQLJ 381
contextos
dependencias de aplicación entre 190
dependencias de base de datos entre 190
establecimiento en aplicaciones DB2 de varias hebras
descripción 187
evitar puntos muertos entre 190
control de la ejecución de sentencias
SQLJ 381
control de relaciones de datos
activadores 47
activadores anteriores 48
activadores posteriores 48
integridad referencial 47
lógica de aplicación 49
control de valores de datos
finalidad 44
lógica de aplicación y tipo de variable 46
restricciones de comprobación de tabla 45
control de valores de datos *(continuación)*
restricciones de integridad referencial 45
restricciones exclusivas 45
tipos de datos 44
vistas con opción check 46
controlador JDBC de DB2 de tipo 2
conexión con fuente de datos
interfaz DriverManager 292
manejo de excepción de SQL 311
seguridad 477
controlador JDBC universal de DB2
códigos de error para errores de controlador 468
conectar con fuente de datos
interfaz DriverManager 294
datos de rastreo, recoger 487
diagnóstico de problemas de JDBC 487
diagnóstico de problemas de SQLJ 487
ejemplo de rastreo 489
estados de SQL para errores de controlador 469
información de cliente ampliada 341
manejo de excepción de SQL 308
métodos definidos solo en 446
propiedades 400
ROWID, JDBC 318
ROWID, SQLJ 379
seguridad 478
seguridad basada en ID de usuario y contraseña 479
seguridad basada solo en el ID de usuario 480
Seguridad Kerberos 482
seguridad mediante ID de usuario cifrado o contraseña cifrada 481
soporte de LOB para JDBC 315
soporte de LOB para SQLJ 376
soporte de redireccionamiento del cliente 339
controlador universal JDBC
instalación 471
conversión de caracteres
codificación de procedimientos almacenados 649, 666
codificación de sentencias de SQL 648
consideraciones sobre programación 646
cuando se ejecuta una aplicación 650
cuándo se produce 650
desbordamiento en la longitud de la serie 666
desbordamiento en la longitud de la serie sobrepasando los tipos de datos 666
durante precompilación y vinculación 650
expansión 653
páginas de códigos soportadas 652
soporte de idioma nacional (NLS) 650
Unicode (UCS2) 668
conversión de página de códigos
sustituciones de caracteres 651

conversión de páginas de códigos de cliente/servidor 650
correlaciones de tipos de datos
entre OLE DB y DB2 245
Java, JDBC y SQL 395
tabla de 245
creación de prototipo del código SQL 42
crear
objetos DB2, JDBC 301
objetos DB2, SQLJ 357
paquetes para aplicaciones compiladas 65
crear API de base de datos
estructura SQLEDBDESC 642
cursores
actualizable
definición 102
en aplicaciones ADO 257
ambiguos 102
bloqueo, consideraciones sobre punto de rescate 687
CLI (interfaz de nivel de llamada)
frente a SQL dinámico incorporado 140
colocación al final de la tabla 107
comportamiento
con sentencia COMMIT 98
después de ROLLBACK TO SAVEPOINT 685
consideraciones sobre COMMIT 98
consideraciones sobre ROLLBACK 98
declaración
en programas de SQL estático 97
desplazable
en aplicaciones ADO 257
filas
actualización 102
supresión 102
finalidad 86, 97
FOR FETCH ONLY 102
IBM OLE DB Provider 245
liberar, comportamiento del bloqueo 98
nombres, REXX 531
paquete invalidado, recuperar filas 98
procesar
con estructura de SQLDA 130
en SQL dinámico 119
resumen 97
programa de ejemplo 103
recuperación de varias filas 97
recuperar filas 97
REXX 539
sólo lectura
consideraciones sobre la unidad de trabajo 98
declaración 97
definición 102
en entornos de bases de datos particionadas 697
SQL dinámico, programa de ejemplo 120
SQL estático, ejemplo 100
tipos 102
cursores *(continuación)*
unidad de trabajo
finalización 98
varios en aplicación 97
WITH HOLD
comportamiento tras COMMIT 98
comportamiento tras ROLLBACK 98
Interfaz XA de X/Open 688
revinculación de paquete durante unidad de trabajo 98

CH

CHAR, tipo de datos
C/C++, conversión 180
REXX 539
char, tipo de datos de C/C++ 180

D

DATE, tipo de datos
C/C++, conversión 180
REXX 539
datos
acceso a especificaciones de Microsoft 13
acceso con WebSphere 563
actualizar 102
coherencia a nivel de transacción 37
confirmación de cambios 38
control de relaciones 46
deshacer cambios con sentencia ROLLBACK 39
entornos de bases de datos particionadas 705
expansión
servidor iSeries 741
servidor OS/390 741
extracción de grandes volúmenes 705
generar para prueba 55
incoherencia 39
recuperados, guardar 105
recuperados anteriormente
actualizar 108
desplazamiento 104
recuperar
con SQL estático 86
segunda vez 105
segunda recuperación 106
suprimir 102
transmisión de grandes volúmenes 729
datos de bytes mixtos
servidor iSeries 741
servidor OS/390 741
datos de prueba
generar 55
datos gráficos
juegos de códigos en chino (tradicional) 656, 659
juegos de códigos en japonés 656, 659
datos gráficos de longitud variable terminados en nulo 180
DB2
obtención de entornos locales 646
DB2 .NET Data Provider 15
DB2 Application Development Client 241
DB2 Connect
niveles de aislamiento 744
proceso de peticiones de interrupción 743
DB2ARXCS.BND, archivo de vinculación para REXX 542
DB2ARXNC.BND, archivo de vinculación para REXX 542
DB2ARXRR.BND, archivo de vinculación para REXX 542
DB2ARXRS.BND, archivo de vinculación para REXX 542
DB2ARXUR.BND, archivo de vinculación para REXX 542
db2bfd, programa de utilidad de descripción de archivos de vinculación 78
DB2CODEPAGE
variable de registro 645
DB2INCLUDE
variable de entorno 196, 219
valor de antememoria de procesador de línea de mandatos 149
DBCLOB, tipo de datos
C/C++, conversión 180
REXX 539
DBCLOB-FILE, tipo de COBOL 210
dbclob_file, tipo de datos de C/C++ 180
DBCLOB-LOCATOR, tipo de COBOL 210
dbclob_locator, tipo de datos de C/C++ 180
DBCS (juego de caracteres de doble byte)
juegos de códigos de japonés y chino tradicional 656
DCL (lenguaje de control de datos)
entornos de sistema principal e iSeries 742
DDL (lenguaje de definición de datos)
en entornos de sistema principal e iSeries 740
rendimiento de SQL dinámico 116
DECIMAL, tipo de datos
C/C++, conversión 180
REXX 539
declaración
variables del lenguaje principal, normas 87
variables en aplicación SQLJ 347
variables en una aplicación JDBC 290
variables indicadoras 91
declaración gráfica de la forma estructurada VARGRAPHIC
C/C++, sintaxis 160
declaraciones de referencias de archivos en REXX 537
DECLARE, sentencias
no soportada en DB2 UDB 750
no soportado en DB2 Connect 750
DECLARE PROCEDURE, sentencia (OS/400) 747

DELETE de posición
SQLJ 363
depurar
programas de aplicación 57
programas FORTRAN 216
desarrollo de aplicaciones
DB2 .NET Data Provider 15
IBM DB2 Development Add-In 4
desbordamiento, numérico 747
desbordamientos por conversión numérica 747
DESCRIBE, sentencia 750
consideraciones sobre Extended UNIX Code 664
no soportado en DB2 Connect 750
proceso de sentencias arbitrarias 135
descriptores de contexto
conexión 140
descriptor 140
entorno 140
sentencia 140
descriptores de contexto de conexión
descripción 140
descriptores de contexto de descriptor
descripción 140
descriptores de contexto de entorno
descripción 140
descriptores de contexto de sentencia
descripción 140
determinación de problemas
guías de aprendizaje 780
información en línea 780
diagnóstico de aplicaciones suspendidas o en bucle 712
diagramas de sintaxis con puntos decimales 782
diseño de aplicación
actualización múltiple, finalidad 672
archivos include, COBOL 194
colocación en antememoria de SQL dinámico 84
consideraciones sobre EUC en japonés y chino tradicional en COBOL 213
consideraciones sobre la conversión de caracteres 646
control de valores de datos 44
conversión de caracteres en sentencias de SQL 648
conversiones de caracteres en los procedimientos almacenados 649
creación de estructura de SQLDA, directrices 131
declarar suficientes entidades SQLVAR 125
definir entorno de prueba 53
descripción de sentencia SELECT 129
desplazarse por datos recuperados anteriormente 104
ejecución de sentencias sin variables 116
ejemplo de Perl 527
estructura de aplicación autónoma 28
guardar peticiones de usuario final 137
lógica en el servidor 49
manejo de errores, directrices 35
diseño de aplicación *(continuación)*
órdenes de clasificación, directrices 637
pasar datos, directrices 134
precompilación 63
proceso de cursor 98
programas de ejemplo 108
prototipo en Perl 525
puntos de código para caracteres especiales 648
recepción de valores NULL 91
recuperación de datos por segunda vez 105
relaciones entre objetos de datos 46
REXX, rutinas de registro 530
sentencias de lista variable, proceso 136
sentencias necesarias 29
seudocódigo 41
soporte de caracteres de doble byte (DBCS) 648
SQL dinámico, finalidad 115
SQL estático, ventajas 84
usuarios simultáneos, tablas temporales declaradas 726
utilización de marcadores de parámetros 137
versiones de paquetes con mismo nombre 72
vinculación 63
DML (lenguaje de manipulación de datos)
entornos de sistema principal e iSeries 741
rendimiento de SQL dinámico 116
documentación
visualizar 767
double, tipo de datos
C and C++ programs 180
DSN en campo SQLERRP
DB2 UDB para OS/390 742
DSS (subsección distribuida)
dirigida 698
DYNAMICRULES, opción de precompilación/vinculación
efectos en SQL dinámico 122

E

EBCDIC
datos de bytes mixtos 741
orden de clasificación 745
ejecutar
SQL en aplicación SQLJ 356
SQL en una aplicación JDBC 300
ejemplos
declaración de localizador de archivos CLOB
FORTRAN 226
declaración de referencias de archivos BLOB
COBOL 206
FORTRAN 226
declaraciones de datos BLOB 161
declaraciones de datos CLOB 161
declaraciones de datos DBCLOB 161
ejemplos *(continuación)*
declarar BLOB
COBOL 204
FORTRAN 225
declarar CLOB
COBOL 204
FORTRAN 225
declarar DBCLOB, COBOL 204
declarar localizador de BLOB, COBOL 205
ejemplo de sección de declaración de SQL para tipos de datos SQL soportados 178
localizador de CLOB 163
marcadores de parámetros, utilizados en búsqueda y actualización 138
miembros de datos de clase en sentencias de SQL 171
programa Perl 527
programa REXX, registrar SQLEXEC, SQLDBS y SQLDB2 530
referencia de archivos CLOB 164
sintaxis, variables de lenguaje principal de tipo carácter, FORTRAN 223
en línea
ayuda, acceso 776
enganche, estado con varias hebras 187
enlace
descripción 70
entero de 64 bits (BIGINT), tipo de datos soportado por DB2 Connect 741
Enterprise Java beans 520
entidades SQLVAR
declarar número suficiente 128
número de variables, declarar 125
entorno, API de
archivo de inclusión
C/C++ 147
COBOL 194
FORTRAN 216
entorno de aplicación, para programación 27
entorno iSeries
acceso a servidores de sistema principal 677
entornos de bases de datos particionadas
aplicaciones suspendidas o en bucle 712
cursores READ ONLY 697
elusión local 698
entorno de prueba, creación 710
errores graves 711
extracción de un gran volumen de datos 705
identificación de la partición que devuelve el error 712
inserciones en almacenamiento intermedio
consideraciones 702
finalidad 699
restricciones 704
manejo de errores 710
optimización de aplicaciones OLTP 697
subsecciones distribuidas, dirigidas 698

entornos de páginas de códigos mixtas
nombres de paquetes 649
entornos de prueba
bases de datos particionadas 710
entornos de sistema principal e iSeries
bloqueo a nivel de fila 746
bloqueo a nivel de página 746
catálogos del sistema 747
consideraciones sobre aplicaciones 739
diferencias en los valores SQLCODE y SQLSTATE 746
estabilidad del cursor 746
lenguaje de control de datos (DCL) 742
lenguaje de definición de datos (DDL) 740
lenguaje de manipulación de datos (DML) 741
niveles de aislamiento de DB2 Connect 744
procedimientos almacenados 747
proceso de peticiones de interrupción 743
series C terminadas en nulo 744
SQLCODE y SQLSTATE autónomos 744
entornos nacionales
cómo los obtiene DB2 646
obtención en programas de aplicación 645
errores
detección en inserción en almacenamiento intermedio 702
manejo en SQLJ 371
errores graves, entornos de bases de datos particionadas 711
espacio GRAPHIC 648
especificación ActiveX Data Object (ADO)
DB2 .NET Data Provider 15
soportado en DB2 13
especificación Objeto de datos remotos (RDO)
soportado en DB2 13
especificaciones de Microsoft
acceso a datos 13
ADO (ActiveX Data Object) 13
MTS (Microsoft Transaction Server) 13
RDO (Remote Data Object) 13
Visual Basic 13
Visual C 13
estabilidad del cursor (CS)
entornos de sistema principal e iSeries 746
estadísticas de catálogo
actualizable por el usuario 42
estado abierto, inserciones en almacenamiento intermedio 702
estado cerrado
inserciones en almacenamiento intermedio 702
estándar FIPS 127-2
declaración de SQLSTATE y SQLCODE como variables del lenguaje principal 110
estándar ISO/ANS SQL92
soporte 744
estándar SQL92
soporte 744
estructura de SQLCA
archivo de inclusión para C/C++ 147
archivos include
aplicaciones COBOL 194
aplicaciones FORTRAN 216
avisos 91
campo SQLCODE 110
campo SQLSTATE 110
campo SQLWARN1 91
definición, programas de ejemplo 108
entornos de bases de datos particionadas
varias estructuras de SQLCA fusionadas 711
notificación de errores 711
requisitos 110
sqlerrd 711
truncamiento de señal 111
varias definiciones 35
varias estructuras fusionadas 711
visión general 110
estructura de SQLDA
asociación con sentencia PREPARE 116
colocación de información sobre sentencia preparada en 116
crear 131
declarar 125
declarar suficientes entidades SQLVAR 128
determinación de tipo de sentencia arbitraria 135
pasar datos 134
pase de bloques de datos 729
preparar sentencias con estructura mínima 126
estructura SQLCHAR
pasar datos con 134
estructura SQLWARN 110
estructuras de datos
declaración 29
definidas por el usuario con varias hebras 188
SQLEDBDESC 642
EUC (Extended UNIX Code)
consideraciones 656
juegos de caracteres 654
EXEC SQL INCLUDE SQLCA 188
expansión de datos
servidor iSeries 741
servidor OS/390 741
expresión CURVAL 723
expresión de lenguaje principal, SQLJ 347, 427
expresión NEXTVAL 723
Extended UNIX Code (EUC)
Archivos DBCLOB 659
consideraciones sobre la clasificación 659
consideraciones sobre UDF (función definida por el usuario) 659
Extended UNIX Code (EUC) *(continuación)*
constantes gráficas 659
conversiones de caracteres, procedimientos almacenados 666
chino (tradicional)
C/C++ 177
consideraciones 658
consideraciones en COBOL 213
FORTRAN 229
juegos de códigos 656
REXX 531
desbordamiento en la conversión de caracteres 666
desbordamiento en la longitud de la serie de caracteres 666
ejemplos de expansión 663
expansión en el servidor 660
expansión en la aplicación 660
Japonés
C/C++ 177
FORTRAN 229
juegos de códigos 656
REXX 531
japonés y chino tradicional
consideraciones en COBOL 213
juegos de caracteres 654
manipulación de datos gráficos 659
páginas de códigos de doble byte 658
páginas de códigos desiguales 660
páginas de códigos mixtas 658
procedimientos almacenados 659
sentencia DESCRIBE 664
tipos de datos de longitud fija 666
tipos de datos de longitud variable 666
validación de parámetros basada en el cliente 663
Extended UNIX Code mixto, consideraciones 658
Extensible Markup Language (XML)
descripción 17
extensiones de archivos de entrada para C/C 146
extensiones de archivos de salida
C/C++ 146
extraer
grandes volúmenes de datos 705

F

filas
captación tras paquete invalidado 98
posicionamiento en tabla 107
recuperación de varias 97
recuperación mediante SQLDA 129
recuperar con cursor 102
segunda recuperación
métodos 105
orden de filas 106
filas, bloqueo
personalización para el rendimiento 729
finalización de transacciones de forma implícita 40

FLOAT, tipo de datos
C/C++, conversión 180
REXX 539
FORCE, mandato
diferencias entre sistemas operativos 742
FORTRAN, lenguaje
archivos de entrada y salida 216
archivos include 216, 219
consideraciones sobre el chino tradicional 229
consideraciones sobre el japonés 229
consideraciones sobre programación 215
declaraciones de datos LOB 225
declaraciones de localizadores de LOB 226
declaraciones de referencias de archivos 226
depurar 216
incorporación de sentencias de SQL 61, 219
juegos de caracteres de varios bytes 229
líneas condicionales 216
líneas de comentario 216
no hay mejoras planificadas 27
no hay soporte para acceso a bases de datos de varias hebras 216
precompilación 216
restricciones 215
sección de declaración de SQL 227
tipos de datos 227
ubicación de los archivos include 219
variables del lenguaje principal
declaración 222
denominación 221
finalidad 221
referencia 219
variables del lenguaje principal numéricas 222
variables indicadoras 224
variables SQLCODE 229
variables SQLSTATE 229
FORTRAN, tipos de datos
BLOB 227
BLOB_FILE 227
BLOB_LOCATOR 227
CLOB 227
CLOB_FILE 227
CLOB_LOCATOR 227
conversión con DB2 227
CHARACTER*n 227
INTEGER*2 227
INTEGER*4 227
REAL*2 227
REAL*4 227
REAL*8 227
fuente de datos
recuperar datos sobre, JDBC 327
fuentes
aplicaciones de SQL incorporado 69
archivos fuente modificados 67
extensiones de archivos SQL 63
extensiones de nombre de archivo 67
fuentes de datos
conectar con
JDBC 291
funciones de preprocesador
y el precompilador de SQL 165
funciones de programación de DB2 18
funciones de tabla OLE DB 241
funciones definidas por el usuario (UDF)
consideración sobre lógica de aplicación 49
juegos de códigos en chino (tradicional) 659
juegos de códigos en japonés 659
visión general 20

G

generador de declaraciones db2dclgn
declaración de variables del lenguaje principal 32
gestor de bases de datos
definición de API, programas de ejemplo 108
gestores de transacciones
COM+ 691
MTS 691
GRAPHIC, tipo de datos
C/C++, conversión 180
REXX 539
GROUP BY, cláusula
orden de clasificación 745
grupo de filas en inserción en almacenamiento intermedio 702
GSS-API
plug-ins de autenticación de GSS-API 632
restricciones 632
guías de aprendizaje 779
resolución y determinación de problemas 780
guías de aprendizaje de DB2 779

H

hebras
IBM OLE DB Provider 242
IBM OLE DB Provider para DB2 241
varias
consideraciones sobre aplicaciones UNIX 189
consideraciones sobre página de códigos 189
consideraciones sobre página de códigos de país/región 189
dependencias de aplicación entre contextos 190
dependencias de base de datos entre contextos 190
evitación de puntos muertos entre contextos 190
problemas potenciales 190
recomendaciones 188
utilización en aplicaciones DB2 187
herramientas
para desarrollo de aplicaciones 3

I

IBM DB2 Development Add-In 4
IBM OLE DB Provider
aplicaciones ADO 256
aplicaciones C/C++
conexiones con fuentes de datos 262
compilar y enlazar en aplicaciones C/C++ 261
conexión de aplicaciones Visual Basic con fuente de datos 257
conexiones con fuentes de datos 255
conjunto de filas de esquema 243
consumidor 241
conversión de datos
de tipos DB2 a tipos OLE DB 248
conversión de datos de OLE DB a tipos DB2 246
cursores 245
cursores en aplicaciones ADO 257
habilitación automática de servicios OLE DB 245
habilitación del soporte de MTS en DB2 263
hebras 242
limitaciones para aplicaciones ADO 257
LOB 243
para DB2
instalación 241
propiedades de OLE DB soportadas 253
proveedor 241
restricciones 250
soporte de OLE DB 250
soporte de transacciones distribuidas MTS y COM 262
soporte para métodos y propiedades ADO 258
tipos de aplicaciones soportadas 242
ID de colección, atributo de
DB2 para iSeries 743
paquete 743
ID de usuario
de dos componentes 575
imprimir
archivos PDF 775
INCLUDE, cláusula 721
incoherencia
datos 39
estados 39
indicaciones horarias
al precompilar 79
información de cliente ampliada
controlador JDBC universal de DB2 341
inserciones, sin inserciones en almacenamiento intermedio 699
inserciones en almacenamiento intermedio
asíncronos 702
consideraciones 702
consideraciones sobre puntos de rescate 699
detección de errores 702
errores de punto muerto 702
estado abierto 702

inserciones en almacenamiento intermedio *(continuación)*
estado cerrado 702
grupo de filas 702
no soportado en CLP 704
notificación de errores 702
opción de vinculación INSERT BUF 699
parcialmente lleno 699
puntos de rescate 687
registros de transacciones 699
restricción de campo largo 704
restricciones 704
SELECT en inserciones en almacenamiento intermedio 702
sentencias que las cierran 699
tamaño del almacenamiento intermedio 699
ventajas 699
violación de clave exclusiva 702
visión general 699
inserciones en almacenamiento intermedio variadas 699
INSERT, sentencia
no soportado en CLP 704
VALUES, cláusula 699
instalación
Centro de información 760, 762, 765
controlador universal JDBC 471
instantáneas de Explain
durante vinculación 77
INTEGER, tipo de datos
C/C++, conversión 180
REXX 539
integridad referencial
consideración sobre las relaciones de datos 47
diferencias entre plataformas 745
interfaces de programación de aplicaciones (API)
consideraciones sobre autorización 53
para establecer contextos entre hebras
sqleAttachToCtx() 187
sqleBeginCtx() 187
sqleDetachFromCtx() 187
sqleEndCtx() 187
sqleGetCurrentCtx() 187
sqleInterruptCtx() 187
sqleSetTypeCtx() 187
restricciones en un entorno XA 688
sintaxis para REXX 542
tipos de 43
usos de 43
visión general de 43
interfaces de programación de aplicaciones (API) de DB2
visión general 7
interfaz DataSource
SQLJ 351
interfaz de nivel de llamada (CLI)
comparación de SQL incorporado y CLI de DB2 140
comparado con SQL incorporado 143
sentencias de SQL soportadas 733
ventajas 141
visión general 140
Interfaz de nivel de llamada de DB2 (CLI de DB2)
comparación con SQL dinámico incorporado 10
visión general 9
interfaz DriverManager
SQLJ 349
interrupción SIGUSR1 111
interrupciones, SIGUSR1 111
invocar
ayuda para mandatos 778
ayuda para mensajes 778
ayuda para sentencias de SQL 779
ISO
estándar 10646 656
estándar 2022 656
iterador
de nombre, SQLJ 358
de posición, SQLJ 361
desplazable, SQLJ 389
obtención de conjuntos de resultados JDBC a partir 373
iterador de conjunto de resultados
declaración pública en archivo separado 373
restricciones referentes a la declaración 361
iterador de posición
aplicación SQLJ 361
pasado como variable, SQLJ 387
iterador desplazable
restricciones para tipos de datos 389
uso en aplicación SQLJ 389
iteradores con nombre en aplicaciones SQLJ 358

J

japonés y chino tradicional, juegos de códigos EUC
consideraciones en COBOL 213
Java
Enterprise Java beans 520
incorporación de sentencias de SQL 61
Java 2 Enterprise Edition
requisitos de base de datos 513
servidor 513
visión general 511
WebSphere Studio, visión general 16
Java 2 Enterprise Edition
contenedores 512
Enterprise Java beans 520
gestión de transacciones 514
requisitos 513
servidor 513
soporte de aplicaciones 511
visión general 511
Java Database Connectivity (JDBC)
visión general 12
Java Naming and Directory Interface (JNDI) 513
JDBC
acceso a paquetes Java para 290
actualizar datos de tablas DB2 303
cerrar conexión con fuente de datos 300
JDBC *(continuación)*
comparación del soporte de controlador de DB2 407
conectar con fuente de datos, interfaz DataSource 297
conjunto de resultados actualizable 335
conjunto de resultados con capacidad de retención 335
conjunto de resultados desplazable 335
controlador JDBC de DB2 de tipo 2
manejo de errores 311
controlador JDBC universal de DB2
manejo de errores 308
controladores soportados 283
correlaciones de tipos de datos 395
diagnosticar problemas, controlador JDBC universal de DB2 487
diferencias, controladores JDBC 459, 466
invocar procedimientos almacenados 306
manejo de un aviso de SQL 312, 313
objetos DataSource
crear y desplegar 338
procedimiento almacenado, recuperar varios conjuntos de resultados 323
recuperar datos de tablas DB2 302, 305
recuperar información sobre parámetros de sentencias 329
recuperar información sobre un conjunto de resultados 326
tipos diferenciados, uso 319
transacción, confirmar 300
transacción, retrotraer 300
transacción distribuida 515
uso de una conexión 299
JDBC (Java database connectivity)
controlador universal JDBC
instalación 471
visión general 12
JDBC ResultSet
controlador JDBC universal de DB2 334
JNDI (Java Naming and Directory Interface) 513
JTA (API de transacciones de Java) 514
JTS (servicio de transacciones de Java) 514
juegos de caracteres
doble byte 654
Extended UNIX Code (EUC) 654
varios bytes, FORTRAN 229
juegos de caracteres de doble byte (DBCS)
consideraciones sobre el chino tradicional 658
consideraciones sobre orden 659
consideraciones sobre programas de aplicación 655
elementos de código 654
juegos de códigos en chino (tradicional) 656
juegos de códigos en japonés 656
páginas de códigos 658

juegos de caracteres de doble byte (DBCS) *(continuación)*
páginas de códigos desiguales 660
juegos de códigos
SQLERRMC, campo de SQLCA 742
juegos de códigos en chino (tradicional)
consideraciones en COBOL 213
consideraciones para C/C++ 177
consideraciones sobre el doble byte 658
Extended UNIX Code
consideraciones 656
consideraciones sobre la conversión 658
FORTRAN 229
REXX, consideraciones 531
UCS2, consideraciones 656
juegos de códigos en japonés
consideraciones para C/C++ 177
Extended UNIX Code, consideraciones 656
FORTRAN 229
REXX 531
UCS2, consideraciones 656

L

LABEL ON, sentencia no soportada 750
LANGLEVEL, opción de precompilación
MIA 180
SAA1 180
variables SQL92E y SQLSTATE o SQLCODE 186, 213, 229, 744
lectura repetible (RR)
método 105
lenguaje C/C++
archivos de entrada 146
archivos de salida 146
archivos include necesarios 147
codificación de caracteres de múltiples bytes 173
consideraciones sobre EUC en chino (tradicional) 177
consideraciones sobre EUC en japonés 177
consideraciones sobre programación 145
declaración de variables del lenguaje principal de gráficos 158
declaración gráfica de la forma estructurada VARGRAPHIC, sintaxis 160
declaraciones de datos LOB 161
declaraciones de localizadores de LOB 163
declaraciones de referencias de archivos 164
depurar 149
expansión de macro 165
FOR BIT DATA 184
incorporación de sentencias de SQL 61
inicialización de variables del lenguaje principal 165
juego de caracteres 146
macro #include, restricciones 149
macros #line, restricciones 149
lenguaje C/C++ *(continuación)*
manejo de series terminadas en nulo 169
miembros de datos de clase 171
opción WCHARTYPE del precompilador 174
operador de calificación, restricción 173
operador de miembro, restricción 173
puntero a tipo de datos 171
secuencias tri-grafo 146
sentencias de SQL incorporado 150
soporte de estructura de sistema principal 166
sqldbchar, tipo de datos 174
tablas de indicadores 168
tipo de datos wchart 174
tipos de datos
para funciones 184
para métodos 184
para procedimientos almacenados 184
soportados 180
tipos de datos soportados 180
variables del lenguaje principal
declaración 154
denominación 153
finalidad 152
variables del lenguaje principal de gráficos 158, 159
variables del lenguaje principal numéricas 154
variables indicadoras 158
variables SQLCODE 186
variables SQLSTATE 186
lenguaje de control de datos (DCL)
entornos de sistema principal e iSeries 742
lenguaje de definición de datos (DDL)
emisión en punto de rescate 685
en entornos de sistema principal e iSeries 740
lenguaje de manipulación de datos (DML)
entornos de sistema principal e iSeries 741
lenguaje de procedimientos SQL 733
lenguaje principal, incorporar sentencias de SQL 61
lenguaje REXX
API
SQLDB2 529
SQLDBS 529
SQLEXEC 529
archivos de vinculación 542
campos decimales de SQLDA
recuperar datos 546
consideraciones sobre programación 529, 530
cursores 539
chino (tradicional) 531
datos LOB 536
declaraciones de localizadores de LOB 537
declaraciones de referencias de archivos LOB 537
lenguaje REXX *(continuación)*
ejecución de aplicaciones 541
identificadores de cursor 531
incorporación de sentencias de SQL 531
inicialización de variables 544
invocar procedimientos almacenados 545
Japonés 531
llamada al CLP de DB2 542
no hay soporte para acceso a bases de datos de varias hebras 531
procedimientos almacenados
consideraciones sobre el servidor 546
llamada 544
visión general 544
registro de rutinas 530
registro de SQLEXEC, SQLDBS y SQLDB2 530
restricciones 530
sentencias de SQL 531
sintaxis de API 542
tipos de datos 539
variables del lenguaje principal
denominación 534
finalidad 533
referencia 534
variables del lenguaje principal LOB, borrado 538
variables indicadoras 534
variables predefinidas 534
liberación
conexiones, aplicaciones CMS 38
LOB (large object), tipos de datos
consideraciones sobre aplicaciones 22
declaraciones de datos en C/C++ 161
declaraciones de localizador en C/C++ 163
IBM OLE DB Provider 243
soportado por DB2 Connect 741
local
elusión 698
lógica de aplicación
activadores 49
control de relaciones de datos 49
control de valores de datos 46
funciones definidas por el usuario 49
procedimientos almacenados 49
servidor 49
long, tipo de datos de C/C++ 180
long int, tipo de datos de C/C++ 180
long long, tipo de datos de C/C++ 180
long long int, tipo de datos de C/C++ 180
LONG VARCHAR, tipo de datos
C/C++, conversión 180
REXX 539
LONG VARGRAPHIC, tipo de datos
C/C++, conversión 180
REXX 539

M

macro #include
restricciones para C/C++ 149

macros #line
  restricciones para C/C++ 149
mandato BIND
  creación de paquetes 71
  opción INSERT BUF 699
mandato BIND PACKAGE
  volver a vincular 81
mandato PREP (PRECOMPILE)
  descripción 67
  ejemplo 67
mandatos
  FORCE 742
manejadores de excepciones
  consideraciones sobre COMMIT y ROLLBACK 111
  finalidad 111
manejadores de interrupciones
  consideraciones sobre COMMIT y ROLLBACK 111
  finalidad 111
manejadores de señales
  con sentencias de SQL 111
  consideraciones sobre COMMIT y ROLLBACK 111
  finalidad 111
  instalación, programas de ejemplo 108
manejo de errores
  aplicaciones en bucle 712
  aplicaciones suspendidas 712
  archivos include
    C/C++ 147
    COBOL 194
    FORTRAN 216
  durante la precompilación 67
  entorno de bases de datos particionadas 710
  entornos de bases de datos particionadas 711
  estructura de SQLCA 711
  estructuras de SQLCA
    utilizar 34
    varias estructuras fusionadas 711
  identificación de la partición de base de datos que devuelve el error 712
  notificación 711
  Perl 527
  precompilador de lenguaje C/C++ 149
  sentencia WHENEVER 35
  SQLCODE 711
manejo de interrupciones con sentencias de SQL 111
manuales de DB2
  imprimir archivos PDF 775
manuales impresos, solicitud 776
marcador de parámetros tipificado 137
marcadores de parámetros
  ejemplo de programación 138
  en proceso de sentencias arbitrarias 135
  entradas SQLVAR 137
  Perl 527
  recuperar información sobre, JDBC 329
  tipificados 137
  uso en SQL dinámico 137
marcadores de parámetros *(continuación)*
  uso en SQLExecDirect 140
memoria
  asignación para páginas de códigos desiguales 660
mensajes de aviso
  truncamiento 91
mensajes de error
  distintivo de condición de aviso 110
  distintivo de condición de error 110
  distintivo de condición de excepción 110
  estructura de SQLCA 110
  estructura SQLWARN 110
  manejo de errores 34
  SQLSTATE 110
métodos
  visión general 20
MIA LANGLEVEL, opción de precompilación 180
Microsoft Component Services (COM+)
  gestor de transacciones 691
Microsoft OLE DB Provider para ODBC
  soporte de OLE DB 250
Microsoft Transaction Server (MTS)
  gestor de transacciones 691
  habilitar soporte en DB2 263
  soporte de transacciones distribuidas MTS y COM 262
  tiempo de espera de transacción 694
Microsoft Transaction Server (MTS), especificaciones
  acceso a datos 13
miembros de datos de clase 171
minusvalías físicas 781
modelo para programación de DB2 41
modificar
  objetos DB2, JDBC 301
  objetos DB2, SQLJ 357
MQSeries
  soporte de aplicaciones 17
MTS (Microsoft Transaction Server), especificaciones
  acceso a datos 13
MTS (Microsoft Transaction Server), soporte de
  gestor de transacciones 691
  habilitación en DB2 263
  tiempo de espera de transacción 694
multibyte, consideraciones sobre
  japonés y chino tradicional, juegos de códigos EUC
    COBOL 213
  juegos de códigos en chino (tradicional)
    C/C 177
    FORTRAN 229
    REXX 531
  juegos de códigos en japonés
    C/C 177
    FORTRAN 229
    REXX 531

N

naturaleza asíncrona de las inserciones en almacenamiento intermedio 702
Net.Data
  visión general 18
nivel de aislamiento
  definir para aplicación JDBC 299
  definir para aplicación SQLJ 354
niveles de aislamiento
  lectura repetible (RR) 105
  plataformas soportadas 744
niveles de versión
  IBM OLE DB Provider para DB2 241
niveles en cascada 745
NLS (national language support)
  conversión de caracteres 650
  datos de bytes mixtos 741
  páginas de códigos 650
NOLINEMACRO, opción de precompilación 149
nombres de paquetes
  entornos de páginas de códigos mixtas 649
nombres de variables de SQLJ
  restricciones 348
notificación de errores 711
NUMERIC, tipo de datos de SQL
  C/C++, conversión 180
  REXX 539
numéricos, tipos de datos
  diferencias entre plataformas 741

O

objetos DataSource, JDBC
  crear y desplegar 338
objetos de SQL
  representación con variables 30
objetos grandes (LOB)
  controlador JDBC universal de DB2 315
  controlador JDBC universal de DB2, SQLJ 376
ODBC (Open Database Connectivity)
  agrupación de conexiones, con MTS y COM+ 695
  herramientas de desarrollo de aplicaciones 15
OLE DB
  conexiones con fuentes de datos mediante IBM OLE DB Provider 255
  conversión de datos
    de OLE DB a tipos DB2 246
    de tipos DB2 a tipos OLE DB 248
  correlaciones de tipos de datos con DB2 245
  funciones de tabla
    visión general 23
  propiedades soportadas 253
  servicios habilitados automáticamente 245
  soportado en DB2 13
  soporte de BLOB 250
  soporte de componentes e interfaces 250
  soporte de conjunto de filas 250
  soporte de mandatos 250
  soporte de sesiones 250
  soporte para ver objetos 250

opción de vinculación EXPLSNAP 77
opción de vinculación FUNCPATH 77
opción de vinculación INSERT BUF
  inserciones en almacenamiento intermedio 699
opción PREP, NOLINEMACRO 149
opción WCHARTYPE del precompilador
  directrices 174
opciones de vinculación
  EXPLSNAP 77
  FUNCPATH 77
  QUERYOPT 77
operador de miembro, restricción en C/C 173
optimizador
  consideraciones sobre SQL estático y dinámico 116
optimizar
  programas de aplicación 57
orden de clasificación definido por el usuario 745, 753

P

páginas de código multibyte
  juegos de códigos en chino (tradicional) 656
  juegos de códigos en japonés 656
páginas de códigos
  asignación de almacenamiento para situaciones desiguales 660
  consideraciones sobre vinculación 77
  conversión
    servidor iSeries 741
    servidor OS/390 741
  conversión de caracteres 650
  conversiones soportadas 652
  cuando se produce conversión de caracteres 650
  entornos nacionales, obtener 645
  manejo de expansiones en el servidor 660
  manejo de expansiones en la aplicación 660
  páginas de códigos de Windows 645
  para ejecución de aplicación 650
  para precompilación y vinculación 650
  situaciones desiguales 653, 660
  soporte de idioma nacional (NLS) 650
  SQLERRMC, campo de SQLCA 742
  variable de registro DB2CODEPAGE 645
páginas de códigos desiguales 660
paquetes
  atributos, por plataforma 743
  crear 65, 71
  descripción 79
  errores de indicación horaria 79
  inoperativo 81
  no válido
    estado 81
  revinculación durante unidad de trabajo
    comportamiento de cursor 98
  soporte de aplicaciones REXX 542
paquetes *(continuación)*
  versiones, privilegios 72
  versiones con mismo nombre 72
parámetro de configuración locktimeout 190
parámetros de COLLECTION 76
parámetros de configuración
  actualización múltiple 676
  locktimeout 190
partición coordinadora, sin inserciones en almacenamiento intermedio 699
particiones de destino
  comportamiento sin inserción en almacenamiento intermedio 699
pase de contextos entre hebras 187
Perl
  conexión con base de datos 525
  consideraciones sobre programación 525
  controladores 525
  devolución de datos 526
  ejemplo de aplicación 527
  especificación DBI (Database Interface) 14
  marcadores de parámetros 527
  no hay soporte para acceso a bases de datos de varias hebras 525
  restricciones 525
  SQLCODE 527
  SQLSTATE 527
peso, definición 637
PICTURE (PIC), cláusula en los tipos de COBOL 210
plug-ins
  de seguridad para autenticación, plug-ins de recuperación de grupos 599, 608, 632
  plug-ins de seguridad
    API 597
    códigos de retorno 590
    desplegar plug-ins de seguridad 581, 583, 584
    limitaciones para el despliegue 737
    mensajes de error 593
    restricciones para plug-ins de GSS-API 633
    secuencia de invocación, orden de invocación para plug-ins 594
    versiones de, crear versiones 634
  seguridad
    nombres, convenios para 574
portabilidad cuando se utiliza CLI en lugar de SQL incorporado 141
precompilación 69
  acceso a servidor de aplicaciones de sistema principal o AS/400 a través de DB2 Connect 69
  acceso a varios servidores 69
  ejemplo 67
  FORTRAN 216
  indicaciones horarias 79
  opción de cursor actualizable 102
  programa de utilidad del marcador 69
  señal de coherencia 79
precompilación *(continuación)*
  soporte de sentencias de SQL dinámico 116
  visión general 67
precompilador
  COBOL 193
  depuración para lenguaje C/C++ 149
  FORTRAN 215
  juego de caracteres para C/C++ 146
  lenguaje C/C++ 173
  número de sección 750
  opción LANGLEVEL SQL92E 744
  opciones 67
  secuencias de tri-grafo para C/C++ 146
  tipos de salida 67
  visión general 61
PREPARE, sentencia
  finalidad 116
  no soportado en DB2 Connect 750
  proceso de sentencias arbitrarias 135
PreparedStatement, métodos de
  sentencias de SQL sin marcadores de parámetros 304
procedimiento almacenado
  recuperar conjuntos de resultados 382
procedimientos almacenados
  aplicaciones REXX 544
  consideración sobre lógica de aplicación 49
  conversión de caracteres 649
  conversión de caracteres, EUC 666
  inicialización
    variables REXX 544
  juegos de códigos en chino (tradicional) 659
  juegos de códigos en japonés 659
  llamada
    JDBC 306
    REXX 544
    SQLJ 370
  plataformas soportadas 747
  recuperar varios conjuntos de resultados en JDBC 323
  visión general 19
procesador de línea de mandatos (CLP)
  llamada desde aplicación REXX 542
  prototipo 42
  sentencias de SQL soportadas 733
  valores de antememoria de variable de entorno DB2INCLUDE 149
programa marcador para la precompilación 69
programa para crear prototipo de estadísticas de catálogo actualizables del usuario 42
programas de aplicación
  activadores, visión general 24
  actualización múltiple con DB2 Connect 749
  CLI de DB2, visión general 9
  COBOL
    sin acceso a bases de datos de varias hebras 193

programas de aplicación *(continuación)*
COBOL *(continuación)*
variables de lenguaje principal, ejemplo 209
conexión con base de datos 36
consideraciones sobre el entorno DBCS 655
DBI de Perl 14
definir entorno de prueba 53
depurar 57
estructura 28
FORTRAN
sin acceso a bases de datos de varias hebras 216
funciones de tabla OLE DB 23
herramientas de usuario final de ODBC 15
interfaces de programación de aplicaciones de DB2 7
Net.Data, visión general 18
optimizar 57
entornos de bases de datos particionadas 697
Perl
sin acceso a bases de datos de varias hebras 525
procedimientos almacenados, visión general 19
requisitos previos 27
REXX
invocar procedimientos almacenados, consideraciones sobre el servidor 546
sin acceso a bases de datos de varias hebras 531
rutinas de lista de salida 112
secuencias, controlar 725
sentencias necesarias 29
SQL estático
códigos de retorno 110
SQL estático, ejemplo 85
SQL incorporado, visión general 8
SQL incorporado para Java (SQLJ), visión general 13
programas de utilidad, API de
archivo de inclusión para aplicaciones C/C++ 147
archivos include
aplicaciones COBOL 194
aplicaciones FORTRAN 216
propiedades
controlador JDBC universal de DB2 400
propiedades de OLE DB soportadas 253
punto de código 637
puntos de rescate
activadores 685
anidados 685
comparación con SQL compuesto 682
consideraciones sobre bloqueo de cursor 687
control 684
crear 684
crear, JDBC 320
crear, SQLJ 354
puntos de rescate *(continuación)*
gestión de transacciones 680
gestores de transacciones XA 688
inserciones en almacenamiento intermedio 687, 699
lenguaje de definición de datos (DDL) 685
liberar, JDBC 320
liberar, SQLJ 354
restricciones 685
SET INTEGRITY, sentencia 685
SQL compuesto atómico 685
puntos muertos
en aplicaciones de varias hebras 190
error en inserción en almacenamiento intermedio 702
evitación de varios contextos 190
evitar en transacciones simultáneas 679
PUT, sentencia no soportada en DB2 Connect 750

Q

QSQ en campo SQLERRP para iSeries 742
queryopt, opción de precompilación/vinculación
consideraciones sobre página de códigos 77

R

rastreo
controlador JDBC universal de DB2, ejemplo 489
rastreos
CLI/ODBC/JDBC 494
REAL, tipo de datos de SQL
C/C++, conversión 180
COBOL 210
FORTRAN 227
REXX 539
REAL*2 FORTRAN, tipo de datos de SQL 227
REAL*4 FORTRAN, tipo de datos de SQL 227
REAL*8 FORTRAN, tipo de datos de SQL 227
recuperación de asignaciones
desbordamientos por conversión numérica 747
recuperar datos
con iterador de nombre, SQLJ 358
con iterador de posición, SQLJ 361
de tablas DB2, JDBC 302, 305
de tablas DB2, SQLJ 357
Perl 526
SQL estático 86
uso de varias instancias de un interador, SQLJ 369
uso de varios iteradores para una tabla DB2 en SQLJ 368
recuperar información de BatchUpdateException 332
recuperar información sobre conjuntos de resultados
JDBC 326
recuperar información sobre marcadores de parámetros
JDBC 329
recuperar información sobre una fuente de datos
JDBC 327
recurso Explain
prototipo 42
REDEFINES, cláusula de COBOL 209
registro de notificación de administración
entornos de bases de datos particionadas 711
registro especial CURRENT EXPLAIN MODE
efecto en SQL dinámico vinculado 73
registro especial CURRENT PATH
efecto en SQL dinámico vinculado 73
registro especial CURRENT QUERY OPTIMIZATION
efecto en SQL dinámico vinculado 73
registros de transacciones, inserciones en almacenamiento intermedio 699
registros especiales
CURRENT EXPLAIN MODE 73
CURRENT PATH 73
CURRENT QUERY OPTIMIZATION 73
remota, unidad de trabajo
finalidad 671
rendimiento
cláusula FOR UPDATE 102
columnas de identidad 716
compilación de sentencias de SQL estático 79
cursores de solo lectura 102, 697
factores que afectan, SQL estático 83
inserciones en almacenamiento intermedio 699
liberación de bloqueos 98
optimización con paquetes 79
secuencias, controlar 725
SQL dinámico 84, 116
SQL estático 84
subsecciones distribuidas, dirigidas 698
rendimiento de aplicación
comparación de objetos de secuencia y columnas de identidad 726
elusión local 698
objetos en secuencia 726
pase de bloques de datos 729
tablas temporales declaradas 726
REORGANIZE TABLE, mandato
páginas de códigos mixtas 658
requisitos del sistema
IBM 0LE DB Provider para DB2 241
resolución de problemas
guías de aprendizaje 780
información en línea 780
restricción para tipos de datos
conjunto de resultados desplazable 335
iterador desplazable 389

restricciones
COBOL 193
en C/C++ 165
FORTRAN 215
IBM OLE DB Provider 250
inserciones en almacenamiento intermedio 704
nombres de variables de SQLJ 348
REXX 530
restricciones de COBOL orientadas a objetos 214
restricciones exclusivas
control de valores de datos 45
restricciones referenciales
control de valores de datos 45
RESULT, variable predefinida de REXX 534
ResultSet
controlador JDBC universal de DB2 334
retrotraer
hasta punto de rescate, JDBC 320
hasta punto de rescate, SQLJ 354
transacción JDBC 300
transacción SQLJ 354
retrotraer cambios 39
ROLLBACK WORK RELEASE, sentencia
no soportado en DB2 Connect 750
ROWID
controlador JDBC universal de DB2 318, 379
ROWID, tipo de datos
soportado por DB2 Connect 741
RUOW
consulte unidad de trabajo remota 671
rutina SQLDB2, registro para REXX 530
rutina SQLDBS, registro para REXX 530
rutinas
automatización de OLE
visión general 23
rutinas Common Language Runtime
tipos de datos de SQL soportados en 238
rutinas de automatización de OLE 23
rutinas de lista de salida
restricciones de uso 112

S

SAA1 LANGLEVEL, opción de precompilación 180
sección de declaración
C/C++ 178
COBOL 200
crear 29
en C/C++ 154
en COBOL 209
FORTRAN 222, 227
normas para sentencias 87
secciones críticas 190
secuencia de identidad 637
secuencias
comparación con columnas de identidad 726
finalidad 723
rendimiento de aplicación 726
secuencias de clasificación
archivos include
C/C 147
COBOL 194
FORTRAN 216
caracteres de varios bytes 637
comparaciones de caracteres 639
comparaciones que no dependen de mayúsculas y minúsculas 640
cuestiones generales 637
ejemplo de orden de clasificación 641
ejemplos 644
especificación 642
función TRANSLATE 640
orden de clasificación EBCDIC y ASCII
descripción 745
ejemplo 641
punto de código 637
secuencia de identidad 637
simulación de orden binario EBCDIC 753
visión general 637
secuencias de clasificación de EBCDIC
ejemplos 644
secuencias de tri-grafo, C/C++ 146
seguridad
controlador JDBC de DB2 de tipo 2 477
controlador JDBC universal de DB2 478
ID de usuario cifrado o contraseña cifrada
controlador JDBC universal de DB2 481
ID de usuario solamente
controlador JDBC universal de DB2 480
ID de usuario y contraseña
controlador JDBC universal de DB2 479
Kerberos
controlador JDBC universal de DB2 482
plug-in
API 606
API, versiones de las API 634
códigos de SQL, estados de SQL correspondientes a 578
depurar, determinación de problemas 578
mensajes de error 593
soporte para ID de usuario de dos componentes 575
plug-ins 569
API 597, 601, 602, 607, 608, 615, 616, 617, 618, 620, 624, 625, 626, 627, 629, 631
API para plug-ins de GSS-API 632
API para plug-ins de ID de usuario/contraseña 608
API para plug-ins de recuperación de grupos 599
API para validar contraseñas 622
bibliotecas de plug-in de seguridad, ubicación 573
seguridad *(continuación)*
plug-ins *(continuación)*
carga de 587
códigos de retorno 590
consideraciones para 32 bits 578
consideraciones para 64 bits 578
denominación 574
desplegar 737
desplegar plug-ins 580, 581
desplegar plug-ins de seguridad 583, 584
limitaciones para el despliegue de plug-ins 737
plug-ins de recuperación de grupos 580
restricciones 633
restricciones para bibliotecas de plug-in de seguridad 588
secuencia de invocación, orden de invocación 594
visión general de la infraestructura del plug-in de seguridad 569
selección completa
consideraciones sobre las inserciones en almacenamiento intermedio 704
SELECT, sentencia
seleccionar de una sentencia de cambio de datos 717
semáforos 190
sentencia BEGIN DECLARE SECTION
crear sección de declaraciones 29
sentencia COMMIT
asociación con cursor 98
finalización de transacción 38
finalización de transacciones 40
sentencia CONNECT
programas de ejemplo 108
valores de SQLCA.SQLERRD 660
sentencia CONNECT RESET 40
sentencia CREATE SEQUENCE
para crear objetos de secuencia 723
sentencia DECLARE CURSOR
adición a una aplicación 36
descripción 97
sentencia END DECLARE SECTION 29
sentencia EXEC SQL INCLUDE 149
sentencia EXECUTE
finalidad 116
sentencia EXECUTE IMMEDIATE
finalidad 116
sentencia FETCH
acceso repetido a datos 105
estructura de SQLDA 129
variables del lenguaje principal 124
sentencia INCLUDE 36
sentencia INCLUDE SQLCA
seudocódigo 34
sentencia INCLUDE SQLDA 36
creación de estructura de SQLDA 131
sentencia RELEASE SAVEPOINT 684
sentencia ROLLBACK
asociación con cursor 98
deshacer los cambios 39
diferencias entre plataformas 742
finalización de transacciones 40
retrotraer cambios 39

sentencia ROLLBACK TO SAVEPOINT
comportamiento de cursor 685
sentencia SAVEPOINT
control de transacciones 684
sentencia SELECT
actualizar datos recuperados 108
asociación con sentencia EXECUTE 116
declarar una SQLDA 125
descripción después de asignar SQLDA 129
inserciones en almacenamiento intermedio 702
lista variable 136
recuperación
varias filas 97
recuperar
datos una segunda vez 105
sentencia DECLARE CURSOR 97
sentencia SET CURRENT, no soportada en DB2 Connect 750
sentencia SET CURRENT PACKAGESET 76
sentencia WHENEVER
manejo de errores 35
sentencias
ACQUIRE, no soportada en DB2 UDB 750
BEGIN DECLARE SECTION 29
CALL, plataformas soportadas 747
CALL USING DESCRIPTOR 747
colocación en antememoria, WebSphere 565
COMMIT 38
COMMIT WORK RELEASE 750
CONNECT 742
CREATE SEQUENCE 723
DB2 Connect
no soportadas 750
soportados 750
DECLARE, no soportada en DB2 UDB 750
DECLARE CURSOR 36
DESCRIBE 750
END DECLARE SECTION 29
INCLUDE 36
INCLUDE SQLCA 34
INCLUDE SQLDA 36
LABEL ON, no soportada en DB2 UDB 750
preparar con estructura de SQLDA mínima 126
RELEASE SAVEPOINT 684
ROLLBACK
diferencias entre plataformas 742
finalización de transacciones 39
tablas temporales declaradas 726
ROLLBACK TO SAVEPOINT 684
SAVEPOINT 684
sentencias de SQL
aplicaciones de la actualización múltiple 673
CONNECT
valores de SQLCA.SQLERRD 660
guardar peticiones de usuario final 137
manejadores de excepciones 111
sentencias de SQL *(continuación)*
manejadores de interrupciones 111
manejadores de señales 111
REXX 531
serializar ejecución 187
sintaxis en COBOL 196
sintaxis en FORTRAN 219
sintaxis en REXX 531
sintaxis para C/C++ 150
sentencias de SQL no ejecutables
DECLARE CURSOR 36
INCLUDE 36
INCLUDE SQLDA 36
señales
truncamiento, estructura de SQLCA 111
señales de coherencia 79
series
opción C, CNULREQD BIND terminada en nulo 744
series gráficas
conversión de caracteres 653
series terminadas en nulo
opción CNULREQD BIND 744
servicio de transacciones de Java (JTS) 514
servicios en tiempo de ejecución
varias hebras
efecto en enganches 187
servidores de sistema principal 677
short, tipo de datos de C/C++ 180
short int, tipo de datos de C/C++ 180
símbolos
sustituciones, restricciones del lenguaje C/C++ 165
sintaxis
declaraciones de indicador LOB, REXX 537
incorporación de sentencias de SQL
REXX 531
sección de declaración
C/C++ 154
COBOL 200
FORTRAN 222
sentencias de SQL incorporado
C/C++ 150
COBOL 196
comentarios, C/C++ 150
comentarios, COBOL 196
comentarios, FORTRAN 219
comentarios, REXX 531
evitar división de línea 150
FORTRAN 219
sustitución de caracteres de espacio en blanco 150
variables del lenguaje principal de tipo carácter 157
SMALLINT, tipo de datos
C/C++, conversión 180
FORTRAN 227
REXX 539
solicitud de manuales de DB2 776
soporte de aplicaciones Java 283
soporte de caracteres multibyte
puntos de código para caracteres especiales 648
soporte de estructura de sistema principal
C/C++ 166
COBOL 206
soporte de idioma nacional (NLS)
conversión de caracteres 650
datos de bytes mixtos 741
páginas de códigos 650
soporte de LOB
fuera de la especificación JDBC 315
localizador de LOB 315
soporte de redireccionamiento del cliente
controlador JDBC universal de DB2 339
soporte de transacciones distribuidas MTS y COM
gestor de transacciones 691
IBM OLE DB Provider 262
SQL, archivo de inclusión
aplicaciones C/C++ 147
aplicaciones COBOL 194
aplicaciones FORTRAN 216
SQL (Structured Query Language)
autorización
API 53
SQL dinámico 51
SQL estático 52
SQL incorporado 50
preparado dinámicamente 140
SQL compuesto
comparación con puntos de rescate 682
soporte de DB2 Connect 748
SQL compuesto NOT ATOMIC
soporte de DB2 Connect 748
SQL dinámico
colocación en antememoria de 84
comparación con SQL estático 83, 116
consideraciones 116
consideraciones sobre autorización 51
cursores, programa de ejemplo 120
declarar SQLDA 125
definición 116
determinación de tipo de sentencia arbitraria 135
efectos de DYNAMICRULES 122
finalidad 115
limitaciones 116
marcadores de parámetros 137
no soportado en DB2 Connect 750
normas de sintaxis 116
PREPARE, sentencia 116, 124
proceso de cursor 119
proceso de cursores 130
rendimiento 116
sentencia DESCRIBE 116, 124
sentencia EXECUTE 116
sentencia EXECUTE IMMEDIATE 116
sentencia FETCH 124
sentencias arbitrarias, proceso de 135
sentencias de SQL soportadas 733
sentencias soportadas 116
soporte de DB2 Connect 739
soporte de Perl 525
suprimir filas 102

SQL dinámico *(continuación)*
vinculación 73
SQL dinámico incorporado 10
SQL estático
autorización 52
consideraciones 116
declaración de variables del lenguaje principal 89
ejemplo de programación de actualización estática 108
Perl, no soportado 525
precompilación, ventajas 79
programa de cursor de ejemplo 100
programa de ejemplo 85
recuperar datos 86
rendimiento 84
soporte de DB2 Connect 739
SQL dinámico
comparación 83, 116
utilización de variables de sistema principal 87
visión general 83
SQL incorporado
autorización 50
COBOL 196
columnas de identidad 716
comentarios
C/C++ 150
COBOL 196
normas 219
comentarios en C/C++ 150
comparado con CLI de DB2 143
ejemplos 61
generadas
columnas 715
generar
valores secuenciales 723
normas
C/C++ 150
FORTRAN 219
normas de sintaxis 61
referencia a variables del lenguaje principal 87, 90
visión general 8, 61
SQL incorporado para Java (SQLJ)
visión general 13
SQL_WCHART_CONVERT, macro del preprocesador 174
SQL1252A, archivo de inclusión
aplicaciones COBOL 194
aplicaciones FORTRAN 216
SQL1252B, archivo de inclusión
aplicaciones COBOL 194
aplicaciones FORTRAN 216
SQLADEF, archivo de inclusión
aplicaciones C/C++ 147
SQLAPREP, archivo de inclusión
aplicaciones C/C++ 147
aplicaciones COBOL 194
aplicaciones FORTRAN 216
SQLCA, archivo de inclusión
aplicaciones C/C++ 147
aplicaciones COBOL 194
aplicaciones FORTRAN 216
SQLCA (área de comunicaciones de SQL)
campo SQLERRP identifica RDBMS 742
SQLCA (área de comunicaciones de SQL) *(continuación)*
consideraciones sobre la utilización de varias hebras 188
informe de errores en inserciones en almacenamiento intermedio 702
inserciones incompletas cuando se producen errores 702
SQLERRMC, campo 742, 748
SQLCA, variable predefinida 534
SQLCA_92, archivo de inclusión
aplicaciones COBOL 194
aplicaciones FORTRAN 216
SQLCA_92, estructura 216
SQLCA_CN, archivo de inclusión 216
SQLCA_CS, archivo de inclusión 216
SQLCLI, archivo de inclusión 147
SQLCLI1, archivo de inclusión 147
SQLCODE
autónoma 744
campo, estructura de SQLCA 110
códigos de error 34
diferencias entre plataformas 746
estructura 110
inclusión de SQLCA 34
notificación de errores 711
SQLCODE -1015
entornos de bases de datos particionadas 711
SQLCODE -1034
entornos de bases de datos particionadas 711
SQLCODE -30081
entornos de bases de datos particionadas 711
SQLCODES, archivo de inclusión
aplicaciones C/C++ 147
aplicaciones COBOL 194
aplicaciones FORTRAN 216
SQLDA
recuperar datos
programas de aplicación REXX 546
SQLDA, archivo de inclusión
aplicaciones C/C++ 147
aplicaciones COBOL 194
aplicaciones FORTRAN 216
SQLDA (área de descriptores de SQL)
consideraciones sobre la utilización de varias hebras 188
SQLDACT, archivo de inclusión 216
SQLDB2, API de REXX 529, 542
sqldbchar, tipo de datos
selección 174
tipo de columna equivalente 180
SQLDBS, API de REXX 529
SQLE819A, archivo de inclusión
aplicaciones C/C++ 147
aplicaciones COBOL 194
aplicaciones FORTRAN 216
SQLE819B, archivo de inclusión
aplicaciones C/C++ 147
aplicaciones COBOL 194
aplicaciones FORTRAN 216
SQLE850A, archivo de inclusión
aplicaciones COBOL 194
aplicaciones FORTRAN 216
SQLE850B, archivo de inclusión
aplicaciones COBOL 194
aplicaciones FORTRAN 216
SQLE859A, archivo de inclusión
aplicaciones C/C++ 147
SQLE859B, archivo de inclusión
aplicaciones C/C++ 147
SQLE932A, archivo de inclusión
aplicaciones C/C++ 147
aplicaciones COBOL 194
aplicaciones FORTRAN 216
SQLE932B, archivo de inclusión
aplicaciones C/C++ 147
aplicaciones COBOL 194
aplicaciones FORTRAN 216
sqleAttachToCtx, API 187
SQLEAU, archivo de inclusión
aplicaciones C/C++ 147
aplicaciones COBOL 194
aplicaciones FORTRAN 216
sqleBeginCtx, API 187
sqleDetachFromCtx, API 187
sqleEndCtx, API 187
sqleGetCurrentCtx, API 187
sqleInterruptCtx, API 187
SQLENV, archivo de inclusión
aplicaciones C/C++ 147
aplicaciones COBOL 194
aplicaciones FORTRAN 216
SQLERRD(1) 653, 660
SQLERRD(2) 653, 660
SQLERRD(3) 688
SQLERRMC, campo de SQLCA 653, 742, 748
SQLERRP, campo de SQLCA 742
sqleSetTypeCtx, API 187
SQLETSD, archivo de inclusión 194
SQLException
manejo 112
SQLEXEC, API de REXX
proceso de sentencias de SQL 531
registro 530
SQL incorporado 529
SQLEXT, archivo de inclusión
aplicaciones CLI 147
sqlint64, tipo de datos de C/C++ 180
SQLISL, variable predefinida 534
SQLJ
acceso a paquetes Java para 346
cerrar conexión con fuente de datos 356
conectar con fuente de datos 349
contexto de ejecución 381
diagnosticar problemas, controlador JDBC universal de DB2 487
expresión de lenguaje principal 347
invocar procedimientos almacenados 370
iterador de posición, pasado como variable 387
manejo de un aviso de SQL 372
tipos diferenciados, uso 381
transacción, confirmar 354
transacción, retrotraer 354
uso de conexión por omisión 353
uso de la interfaz DataSource 351

SQLJ *(continuación)*
uso de la interfaz DriverManager 349
varias instancias de un iterador 369
varios iteradores para una tabla 368
SQLJ, cláusula de asignación 437
SQLJ, cláusula de contexto 434
SQLJ, cláusula de conversión a iterador 438
SQLJ, cláusula de declaración de conexión 430, 431
SQLJ, cláusula de sentencia 434
SQLJ, cláusula ejecutable 433
SQLJ, cláusula implements 428
SQLJ, cláusula SET-TRANSACTION 436
SQLJ, cláusula with 429
SQLJ, expresión de lenguaje principal 427
sqlj.runtime.ConnectionContext
métodos 438
sqlj.runtime.ExecutionContext
métodos 438
sqlj.runtime.ForUpdate
métodos 438
sqlj.runtime.NamedIterator
métodos 438
sqlj.runtime.PositionedIterator
métodos 438
sqlj.runtime.ResultSetIterator
métodos 438
sqlj.runtime.Scrollable
métodos 438
SQLJACB, archivo de inclusión
aplicaciones C/C++ 147
SQLMON, archivo de inclusión
aplicaciones COBOL 194
aplicaciones FORTRAN 216
para aplicaciones C/C++ 147
SQLMONCT, archivo de inclusión 194
SQLMSG, variable predefinida 534
SQLRDAT, variable predefinida 534
SQLRIDA, variable predefinida 534
SQLRODA, variable predefinida 534
SQLSTATE
autónoma 744
códigos emitidos por el controlador JDBC universal de DB2 469
diferencias 746
en CLI 140
SQLSTATE, archivo de inclusión
aplicaciones C/C++ 147
aplicaciones COBOL 194
aplicaciones FORTRAN 216
SQLSYSTM, archivo de inclusión 147
SQLUDF, archivo de inclusión
aplicaciones C/C++ 147
SQLUTBCQ, archivo de inclusión 194
SQLUTBSQ, archivo de inclusión 194
SQLUTIL, archivo de inclusión
aplicaciones C/C++ 147
aplicaciones COBOL 194
aplicaciones FORTRAN 216
SQLUV, archivo de inclusión
aplicaciones C/C++ 147
SQLUVEND, archivo de inclusión 147
SQLXA, archivo de inclusión
aplicaciones C/C++ 147
Structured Query Language (SQL)
sentencias soportadas
Interfaz de nivel de llamada (CLI) 733
lenguaje de procedimientos SQL 733
Procesador de línea de mandatos (CLP) 733
SQL dinámico 733
subsección distribuida (DSS)
dirigida 698
sucesos asíncronos 187
SYSIBM.SYSPROCEDURES, catálogo (OS/390) 747

T

tablas
autorreferentes 745
colocación del cursor al final 107
columnas de identidad 716
columnas generadas 715
confirmación de cambios 38
no conectado inicialmente, creación en punto de rescate 685
nombres
resolver no calificados 76
recuperar filas, ejemplo 103
resolución de nombres no calificados 76
restricciones de comprobación
control de valores de datos 45
temporal declarada
creación en punto de rescate 685
creación fuera de punto de rescate 685
temporales
declarada 726
tablas de indicadores
C/C++ 168
soporte de COBOL 208
tablas de prueba, creación 54
tablas temporales
declarada 726
tablas temporales declaradas
finalidad 726
sentencia ROLLBACK 726
territorio, campo SQLERRMC de SQLCA 742
TIME, tipo de datos
C/C++, conversión 180
FORTRAN 227
REXX 539
TIMESTAMP, tipo de datos
C/C++, conversión 180
FORTRAN 227
REXX 539
tipo de controlador JDBC
definición 283
Tipo de datos BIGINT
en SQL estático 93
tipo de datos BLOB
FORTRAN 227
SQL estático 93
tipo de datos BLOB_FILE
FORTRAN 227
tipo de datos BLOB FORTRAN 227
tipo de datos BLOB_LOCATOR
FORTRAN 227
tipo de datos CLOB_FILE
FORTRAN 227
tipo de datos CLOB FORTRAN 227
tipo de datos CLOB_LOCATOR
FORTRAN 227
tipo de datos CHAR
COBOL 210
FORTRAN 227
variables indicadoras 93
tipo de datos CHARACTER*n
FORTRAN 227
tipo de datos DATE 93
COBOL 210
FORTRAN 227
tipo de datos DBCLOB
COBOL 210
en programas de SQL estático 93
juegos de códigos en chino (tradicional) 659
juegos de códigos en japonés 659
tipo de datos de SQL BIGINT
FORTRAN 227
soportado por DB2 Connect 741
tipo de datos DECIMAL
COBOL 210
en SQL estático 93
FORTRAN 227
tipo de datos DOUBLE 93
tipo de datos FLOAT 93
COBOL 210
FORTRAN 227
tipo de datos FOR BIT DATA, C/C++ 184
tipo de datos GRAPHIC
COBOL 210
FORTRAN, no soportado 227
selección 174
tipo de datos INTEGER 93
COBOL 210
FORTRAN 227
tipo de datos INTEGER*2
FORTRAN 227
tipo de datos INTEGER*4
FORTRAN 227
tipo de datos integer de 64 bits
soportado por DB2 Connect 741
tipo de datos LONG VARCHAR
COBOL 210
en programas de SQL estático 93
FORTRAN 227
tipo de datos LONG VARGRAPHIC
COBOL 210
en programas de SQL estático 93
FORTRAN 227
tipo de datos NUMERIC de SQL
COBOL 210
FORTRAN 227
tipo de datos REAL de SQL
lista 93
tipo de datos SMALLINT
COBOL 210
sentencia CREATE TABLE 93
tipo de datos terminado en nulo de C/C++ 180

tipo de datos TIME
COBOL 210
en sentencia CREATE TABLE 93
tipo de datos TIMESTAMP
COBOL 210
descripción 93
tipo de datos VARCHAR
C o C++ 184
COBOL 210
en columnas de tablas 93
tipo de datos VARGRAPHIC
COBOL 210
lista 93
tipos de datos
BINARY
COBOL 213
C/C++ 180
C/C++, conversión 180
CLOB en C/C++ 184
COBOL 210
consideraciones sobre Extended UNIX Code 666
control de valores de datos 44
conversión
entre DB2 y COBOL 210
entre DB2 y FORTRAN 227
entre DB2 y REXX 539
conversión entre DB2 y C/C++ 180
DATALINK
variable del lenguaje principal, restricción 227
DECIMAL
FORTRAN 227
desbordamiento en la conversión de caracteres 666
descripción 29
FOR BIT DATA
COBOL 213
FOR BIT DATA en C/C++ 184
FORTRAN 227
lenguaje de sistema principal y correspondencias en DB2 93
miembros de datos de clase, declarar en C/C++ 171
numéricos
diferencias entre plataformas 741
puntero a, declarar en C/C++ 171
ROWID
soportado por DB2 Connect 741
selección de tipos gráficos 174
soportados 93
COBOL, normas 210
FORTRAN, normas 227
temas sobre compatibilidad 93
VARCHAR en C/C++ 184
tipos de datos COMP, en COBOL 213
tipos de datos COMP-1, en COBOL 210
tipos de datos COMP-3, en COBOL 210
tipos de datos COMP-4, en COBOL 213
tipos de datos COMP-5, en COBOL 210
tipos de datos de REXX 539
tipos de datos de SQL
BIGINT 93
BLOB 93
C/C++, conversión 180
CLOB 93
COBOL 210
tipos de datos de SQL *(continuación)*
CHAR 93
DATE 93
DBCLOB 93
DECIMAL 93
FLOAT 93
FORTRAN 227
INTEGER 93
LONG VARCHAR 93
LONG VARGRAPHIC 93
REAL 93
REXX 539
SMALLINT 93
TIME 93
TIMESTAMP 93
VARCHAR 93
VARGRAPHIC 93
tipos de datos LOB (large object)
consideraciones sobre aplicaciones 22
declaraciones de datos en C/C++ 161
declaraciones de localizador en C/C++ 163
IBM OLE DB Provider 243
soportado por DB2 Connect 741
tipos de datos y cursores desplazables
restricciones 335, 389
tipos definidos por el usuario (UDT)
consideraciones sobre aplicaciones 22
soportado por DB2 Connect 741
tipos diferenciados
en aplicaciones JDBC 319
en aplicaciones SQLJ 381
soportado por DB2 Connect 741
tipos estructurados
no soportados en DB2 Connect 741
transacciones
codificación 37
coherencia de datos 37
confirmación de trabajo 38
débilmente acopladas 693
deshacer cambios con sentencia ROLLBACK 39
finalización
sentencia COMMIT 40
sentencia CONNECT RESET 40
sentencia ROLLBACK 40
finalización implícita 40
puntos de rescate 680
simultánea
evitar puntos muertos 679
finalidad 678
problemas potenciales 679
tiempo de espera, con MTS y COM+ 694
transacciones, supervisores del proceso
Interfaz XA de X/Open 688
transacciones distribuidas
ejemplo 515
transacciones simultáneas
evitar puntos muertos 679
finalidad 678
problemas potenciales 679
transferencia de datos
actualizar 108
transmisión de grandes volúmenes de datos 729
truncamiento
variables del lenguaje principal 91
variables indicadoras 91

U

ubicación de archivos include, FORTRAN 219
Unicode (UCS-2)
consideraciones sobre UDF (función definida por el usuario) 659
conversión de caracteres 668
desbordamiento en la conversión de caracteres 666
juegos de códigos en chino (tradicional) 656
juegos de códigos en japonés 656
unidad de trabajo distribuida 672
unidades de trabajo (UOW)
codificación 37
consideraciones sobre cursor 98
finalización, comportamiento del cursor 98
remota 671
UPDATE de posición
SQLJ 363
USAGE, cláusula en los tipos de COBOL 210

V

validación de parámetros basada en el cliente 663
valor nulo, SQL
recepción por variable de indicador 91
Valores de SQLCA.SQLERRD en CONNECT 660
valores secuenciales, véase secuencias 723
VARCHAR, tipo de datos
C/C++, conversión 180
formato estructurado, C/C++ 180
FORTRAN 227
REXX 539
VARGRAPHIC, tipo de datos
C/C++, conversión 180
FORTRAN 227
REXX 539
variables
declaración 29
interacción con gestor de bases de datos 29
representación de objetos de SQL 30
REXX, predefinido 534
SQLCODE 186, 213, 229
SQLSTATE 186, 213, 229
variables de entorno
DB2INCLUDE 149, 219
variables del lenguaje principal
COBOL, tipos de datos 210
codificación de caracteres de múltiples bytes 173
consideradas como globales por el precompilador para un módulo en C/C++ 153

variables del lenguaje principal
(continuación)
declaración
C/C++ 154
COBOL 200
como puntero a tipo de datos 171
con el generador de declaraciones db2dclgn 32
FORTRAN 222
localizador de LOB, COBOL 205
normas 87
programas de ejemplo 108
programas de SQL estático 89
utilización de sentencia de lista de variables 136
declaraciones de datos LOB
C/C++ 161
COBOL 204
FORTRAN 225
REXX 536
declaraciones de localizadores de LOB
C/C++ 163
FORTRAN 226
REXX 537
declaraciones de referencias de archivos
C/C++ 164
COBOL 206
FORTRAN 226
REXX 537
declarar
gráficas, COBOL 203
definición 87
definición para utilizar con columnas 33
denominación
C/C++ 153
COBOL 199
FORTRAN 221
REXX 534
en sentencia de lenguaje de sistema principal 87
en sentencia de SQL 87
en SQL dinámico 116
finalidad 152
FORTRAN 221
gráfico
C/C++ 158
FORTRAN 229
inicializar en C/C++ 165
LOB, borrar en REXX 538
miembros de datos de clase en C/C++ 171
no soportado en Perl 526
opción WCHARTYPE del precompilador 174
pase de bloques de datos 729
referencia
C/C++ 152
COBOL 198
FORTRAN 219
REXX 534
referencia desde SQL 87, 90
relacionadas con sentencia de SQL 33
restricción de DATALINK 227
REXX 533
variables del lenguaje principal
(continuación)
selección de tipos de datos gráficos 174
series terminadas en nulo, manejo en C/C++ 169
sintaxis para caracteres de longitud fija, COBOL 201
SQL estático 87
truncamiento 91
variables del lenguaje principal de gráficos
C/C++ 159
variables del lenguaje principal de tipo carácter
fijas y terminadas en nulo de C/C++ 156
FORTRAN 223
longitud variable en C/C++ 157
variables del lenguaje principal numéricas
C/C++ 154
COBOL 200
FORTRAN 222
variables gráficas de lenguaje principal
COBOL 203
variables indicadoras
C/C++ 158
COBOL 204
declaración 91
durante INSERT o UPDATE 91
finalidad 91
FORTRAN 224
REXX 534
truncamiento 91
utilización en columnas que pueden tener valores null 95
varios conjuntos de resultados
recuperar de un procedimiento almacenado 382
vinculación
consideraciones 77
diferir 78
opciones 71
programa de utilidad de descripción de archivos de vinculación, db2bfd 78
sentencias dinámicas 73
visión general 71
violación de clave exclusiva, inserciones en almacenamiento intermedio 702
vistas
catálogos del sistema 747
control de valores de datos 46
vistas de prueba, creación 54
vistas del catálogo del sistema
programa de utilidad de creación de prototipos 42
Visual Basic
aplicaciones, conectar con fuente de datos 257
consideraciones sobre cursor 257
soportado en DB2 13
soporte de control de datos 257
Visual C
soportado en DB2 13
volver a vincular
descripción 81
mandato REBIND PACKAGE 81

W

wchar_t, tipo de datos
selección 174
WCHARTYPE, opción de precompilador
tipos de datos disponibles con la opción NOCONVERT 180
WebSphere
acceso a datos de la empresa 563
agrupación de conexiones
beneficios 564
finalidad 563
colocación en antememoria de sentencias 565
fuentes de datos 563
WebSphere Studio 16
Windows
páginas de códigos 645
variable de registro DB2CODEPAGE 645

X

XA, interfaz
aplicación de una soba hebra 688
aplicación de varias hebras 688
características del proceso de transacciones 688
cursores declarados WITH HOLD 688
DISCONNECT 688
enlace de aplicaciones 691
entorno CICS 688
entorno XA 688
finalidad 688
puntos de rescate 688
RELEASE no soportado 688
restricciones de la API 688
sentencia COMMIT 688
sentencia ROLLBACK 688
SQL CONNECT 688
transacciones 688
XASerialize 688
XML Extender
visión general 17

Cómo ponerse en contacto con IBM

En los EE.UU., puede ponerse en contacto con IBM llamando a uno de los siguientes números:

- 1-800-IBM-SERV (1-800-426-7378) para servicio al cliente
- 1-888-426-4343 para obtener información sobre las opciones de servicio técnico disponibles
- 1-800-IBM-4YOU (426-4968) para marketing y ventas de DB2

En Canadá, puede ponerse en contacto con IBM llamando a uno de los siguientes números:

- 1-800-IBM-SERV (1-800-426-7378) para servicio al cliente
- 1-800-465-9600 para obtener información sobre las opciones de servicio técnico disponibles
- 1-800-IBM-4YOU (1-800-426-4968) para marketing y ventas de DB2

Para localizar una oficina de IBM en su país o región, consulte IBM Directory of Worldwide Contacts en el sitio Web http://www.ibm.com/planetwide

Información sobre productos

La información relacionada con productos DB2 Universal Database se encuentra disponible por teléfono o a través de la World Wide Web en el sitio http://www.ibm.com/software/data/db2/udb

Este sitio contiene la información más reciente sobre la biblioteca técnica, pedidos de manuales, descargas de productos, grupos de noticias, FixPaks, novedades y enlaces con recursos de la Web.

Si vive en los EE.UU., puede llamar a uno de los números siguientes:

- 1-800-IBM-CALL (1-800-426-2255) para solicitar productos u obtener información general.
- 1-800-879-2755 para solicitar publicaciones.

Para obtener información sobre cómo ponerse en contacto con IBM desde fuera de los EE.UU., vaya a la página IBM Worldwide en el sitio www.ibm.com/planetwide

SC10-3723-01

Spine information:

IBM® DB2 Universal Database™ Programación de aplicaciones de cliente Versión 8.2