IBM® DB2 Universal Database™

Guía para el desarrollo de aplicaciones: Aplicaciones de servidor de programación

Versión 8.2

SC10-3724-01

IBM® DB2 Universal Database™

Guía para el desarrollo de aplicaciones: Aplicaciones de servidor de programación

Versión 8.2

SC10-3724-01

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 Server Applications, Version 8.2*, (SC09-4827-01).

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).

Contenido

Acerca de este manual vii

Parte 1. Rutinas 1

Capítulo 1. Introducción a las rutinas . . 3
Rutinas en el desarrollo de aplicaciones 3
Tipos de rutinas (procedimientos, funciones, métodos) 5
Rutinas definidas por el usuario 10
Comparación de procedimientos, funciones y métodos 13
Procedimientos 13
Funciones escalares definidas por el usuario . . 15
Funciones escalares definidas por el usuario . . 17
Métodos 18

Capítulo 2. Desarrollo de las rutinas . . 21
Lenguajes de programación de rutinas soportados 21
Prácticas óptimas para el desarrollo de rutinas . . 24
Consideraciones sobre el rendimiento para crear rutinas 24
Consideraciones sobre seguridad para las rutinas 27
Consideraciones sobre la gestión de bibliotecas y clases 30
Restricciones del uso de rutinas 32
Creación de rutinas en la base de datos 35
Escritura de rutinas. 36
Autorizaciones y enlace de rutinas que contienen SQL 38
Depuración de rutinas. 42
Conflictos de datos cuando los procedimientos leen o graban en tablas 44
Características de los procedimientos 46
Modalidades de parámetros de procedimiento. . 46
Conjuntos de resultados de procedimiento . . . 47
Manejo de los parámetros en los procedimientos de PROGRAM TYPE MAIN o PROGRAM TYPE SUB 56
Características de las UDF y los métodos 58
Áreas reutilizables para UDF y métodos. . . . 58
Áreas reutilizables en sistemas operativos de 32 bits y 64 bits 61
Modelo de proceso de los métodos y las funciones escalares 62
Funciones de tabla definidas por el usuario. . . . 63
Funciones de tabla definidas por el usuario. . . 63
Modelo de proceso de las funciones de tabla . . 63
Modelo de ejecución de funciones de tabla para Java 65

Capítulo 3. Rutinas de SQL 67
SQL Procedural Language (SQL PL) en DB2 . . . 67
Sentencias CREATE para las rutinas de SQL . . . 68
Niveles de acceso de SQL en rutinas de SQL . . . 69
SQL dinámico en las rutinas de SQL 69
Procedimientos de SQL/ SQL PL 71
Consideraciones de diseño para los procedimientos de SQL 71
Creación de procedimientos de SQL desde la línea de mandatos 72
Parámetros de los procedimientos de SQL . . . 74
Variables en los procedimientos de SQL (sentencias DECLARE, DEFAULT, SET) 75
Bloques compuestos y ámbito de las variables en los procedimientos de SQL 76
Devolución de mensajes de error desde procedimientos de SQL 77
Manejadores de condiciones en los procedimientos de SQL 77
Mejora del rendimiento de los procedimientos de SQL 82
Funciones de tabla de SQL 88
Funciones de tabla de SQL que modifican datos de SQL 88
Auditoría mediante funciones de tabla de SQL 90

Capítulo 4. Rutinas externas 95
Estilos de parámetros para rutinas externas. . . . 95
Sintaxis para pasar argumentos a rutinas escritas en C/C++, OLE o COBOL 97
SQL en rutinas externas 110
Efecto de la opción de vinculación DYNAMICRULES en SQL dinámico. 113
Rutinas de ejecución en el lenguaje común .NET 116
Rutinas de ejecución en el lenguaje común (CLR) 116
Creación de rutinas CLR. 117
Tipos de datos de SQL soportados para el Proveedor de datos DB2 .NET. 120
Parámetros de las rutinas CLR 121
Devolución de conjuntos de resultados desde procedimientos CLR 125
Restricciones de las rutinas CLR 126
Errores relacionados con rutinas CLR 128
Ejemplos de procedimientos CLR en C# . . . 130
Ejemplos de procedimientos CLR en Visual Basic 142
Ejemplos de funciones CLR definidas por el usuario en C# 152
Ejemplos de funciones CLR definidas por el usuario en Visual Basic 158
Rutinas C/C++. 164
Rutinas C/C++. 164
Archivo de inclusión para rutinas C/C++ (sqludf.h). 168
Tipos de datos de SQL soportados en C/C++ 168
Manejo de tipos de datos de SQL en rutinas C/C++ 171

Variables gráficas del lenguaje principal en rutinas C/C++ 179
Decoración de tipos de C++ 179
Rutinas Java 181
Rutinas Java 181
Tipos de datos de SQL soportados en Java . . 185
Dónde se deben colocar las clases de Java . . . 186
Actualización de rutinas Java (procedimientos almacenados, UDF y métodos) para la ejecución. 187
Administración de los archivos JAR en el servidor de bases de datos 187
Contextos de conexión en rutinas SQLJ 189
Depuración de procedimientos almacenados en Java 189
Rutinas de automatización de OLE 194
Diseño de rutinas de automatización de OLE 194
Creación de rutinas de automatización de OLE 195
Consideraciones sobre instancias de objetos y el área reutilizable y las rutinas de OLE 196
Tipos de datos de SQL soportados en la automatización de OLE 197
Rutinas de automatización de OLE en BASIC y C++ 198
Funciones de tabla de OLE DB definidas por el usuario 201
Funciones de tabla de OLE DB definidas por el usuario 201
Creación de una UDF para tablas OLE DB . . 203
Nombres de conjunto de filas completamente calificados 205
Tipos de datos de SQL soportados en OLE DB 206

Capítulo 5. Invocación de rutinas . . . 209

Invocación de la rutina 209
Vías de acceso y nombres de rutina 211
Invocaciones de rutinas anidadas 213
Invocación de rutinas de 32 bits en un servidor de bases de datos de 64 bits 213
Consideraciones sobre la página de códigos de las rutinas 214
Invocación de procedimientos 215
Referencias a procedimientos 215
Selección de procedimiento 216
Llamada a procedimientos desde aplicaciones o rutinas externas 217
Llamada a procedimientos desde activadores o rutinas de SQL 219
Llamada a procedimientos desde el Procesador de línea de mandatos (CLP) 222
Invocación de funciones y métodos 224
Referencias a funciones 224
Selección de función 225
Tipos diferenciados como parámetros de UDF o método 227
Valores de LOB como parámetros de UDF. . . 228
Invocación de funciones escalares o métodos 229
Invocación de funciones de tabla definidas por el usuario 230

Parte 2. Objetos grandes, tipos diferenciados definidos por el usuario y activadores 233

Capítulo 6. Objetos grandes 235

Uso de objetos grandes 235
Localizadores de objetos grandes 236
Recuperación de un valor LOB con un localizador de LOB 238
Aplazamiento de la evaluación de las expresiones LOB 240
Variables de referencia a archivos de objetos grandes 242
Grabación de datos de una columna CLOB en un archivo de texto 244
Inserción de datos de un archivo de texto en una columna CLOB 245

Capítulo 7. Tipos diferenciados definidos por el usuario 247

Tipos definidos por el usuario 247
Tipos diferenciados definidos por el usuario . . . 247
Tipificación estricta en tipos diferenciados definidos por el usuario 249
Creación de tipos diferenciados 250
Creación de tablas con columnas basadas en tipos diferenciados 251
Eliminación de tipos definidos por el usuario . . 252
Creación de tipos diferenciados basados en la moneda 253
Creación de un tipo diferenciado para formularios de solicitud de trabajo cumplimentados 254
Creación de tablas para hacer un seguimiento de las ventas internacionales 255
Creación de una tabla para almacenar formularios de solicitud de trabajo cumplimentados 255
Manipulación de tipos diferenciados. 256
Manipulación de tipos diferenciados. 256
Conversión del tipo de datos entre tipos diferenciados 257
Realización de comparaciones que implican a tipos diferenciados 258
Realización de comparaciones entre tipos diferenciados y constantes 259
Realización de asignaciones que implican a tipos diferenciados en SQL incorporado . . . 260
Realización de asignaciones que implican a tipos diferenciados en SQL dinámico 260
Realización de asignaciones que implican a distintos tipos diferenciados 261
Realización de operaciones UNION sobre columnas con tipo diferenciado 262
Definición de UDF con fuente para tipos diferenciados 262

Capítulo 8. Tipos estructurados definidos por el usuario 265

Tipos estructurados definidos por el usuario . . . 266
Creación de tipos estructurados 266

Almacenamiento de instancias de tipos estructurados 267
Posibilidad de creación de instancias en tipos estructurados 268
Jerarquías de tipos estructurados 268
Creación de una jerarquía de tipos estructurados 270
Definición del comportamiento de los tipos estructurados 271
Despacho dinámico de métodos 272
Rutinas generadas por el sistema para tipos estructurados 274
Funciones de comparación y de conversión del tipo de datos para tipos estructurados 274
Funciones de constructor para tipos estructurados 274
Métodos de mutador para tipos estructurados 275
Métodos de observador para tipos estructurados 275
Tablas con tipo 276
Tablas con tipo 276
Creación de tablas con tipo 276
Eliminación de tablas con tipo 279
Posibilidad de sustitución en las tablas con tipo 280
Almacenamiento de objetos en filas de tablas con tipo 281
Definición de identificadores de objetos generados por el sistema 283
Definición de restricciones sobre columnas de identificador de objetos 285
Tipos de referencia 286
Vistas con tipo 290
Vistas con tipo 290
Creación de vistas con tipo 291
Modificación de vistas con tipo 293
Eliminación de vistas con tipo 293
Consulta de tablas con tipo y vistas con tipo . . . 294
Emisión de consultas para deshacer referencias 294
Devolución de objetos de un tipo determinado mediante ONLY 296
Restricción de los tipos devueltos mediante un predicado TYPE 296
Devolución de todos los tipos posibles mediante OUTER 297
Tipos estructurados como tipos de columna . . . 298
Almacenamiento de objetos de tipo estructurado en columnas de tablas 298
Inserción en columnas de atributos de tipo estructurado 300
Definición y modificación de tablas con columnas de tipo estructurado 301
Definición de tipos con atributos de tipo estructurado 301
Inserción de filas que contienen valores de tipo estructurado 302
Modificación de valores de tipo estructurado en columnas 303
Funciones de transformación y grupos de transformación 306
Funciones de transformación y grupos de transformación 306
Recomendaciones para denominar grupos de transformación 307
Especificación de grupos de transformación . . 308
Creación de la correlación con el programa del lenguaje principal 310
Correlaciones del programa del lenguaje principal con funciones de transformación. . . 310
Transformaciones de función 311
Implementación de transformaciones de función utilizando rutinas incorporadas al SQL 313
Pase de parámetros de tipo estructurado a rutinas externas 314
Transformaciones de cliente 316
Implementación de transformaciones de cliente mediante UDF externas 319
Implementación de transformaciones de cliente para enlazar desde un cliente mediante UDF externas 319
Consideraciones sobre la conversión de datos 320
Requisitos de las funciones de transformación 321
Recuperación de datos de subtipo de DB2 . . . 322
Devolución de datos de subtipo a DB2 325
Variables del lenguaje principal de tipo estructurado 329
Declaración de variables del lenguaje principal de tipo estructurado 329
Descripción de un tipo estructurado 329

Capítulo 9. Activadores 331
Activadores en el desarrollo de aplicaciones . . . 331
Activadores INSERT, UPDATE y DELETE 335
Interacciones de los activadores con las restricciones referenciales 336
Activadores INSTEAD OF 336
Directrices para la creación de activadores 337
Creación de activadores 338
Granularidad del activador 339
Tiempo de activación del activador 340
Variables de transición 343
Tablas de transición 344
Acción activada 346
Acción activada 346
Acciones activadas calificadas por condiciones 346
Acción activada compuesta por sentencias de SQL 347
Acción activada que contiene una referencia de función o procedimiento. 348
Activadores múltiples 350
Sinergia entre activadores, restricciones y rutinas 351
Extracción de información de los UDT, las UDF y los LOB con activadores 351
Prevención de operaciones sobre tablas mediante activadores 352
Definición de reglas empresariales mediante activadores 353
Definición de acciones mediante activadores . . 353

Parte 3. Apéndices 355

Apéndice A. Rutinas DB2GENERAL 357
Rutinas DB2GENERAL 357
UDF DB2GENERAL 358

Tipos de datos de SQL soportados en rutinas DB2GENERAL 360
Clases de Java para rutinas DB2GENERAL . . . 362
Clases de Java para rutinas DB2GENERAL . . 362
Clase DB2GENERAL de Java: COM.IBM.db2.app.StoredProc 362
Clase DB2GENERAL de Java: COM.IBM.db2.app.UDF 364
Clase DB2GENERAL de Java: COM.IBM.db2.app.Lob 366
Clase DB2GENERAL de Java: COM.IBM.db2.app.Blob 367
Clase DB2GENERAL de Java: COM.IBM.db2.app.Clob 367

Apéndice B. Procedimientos COBOL 369
Procedimientos COBOL 369
Tipos de datos de SQL soportados en COBOL . . 371

Apéndice C. Información técnica sobre DB2 Universal Database 375
Documentación y ayuda de DB2 375
Actualizaciones de la documentación de DB2 375
Centro de información de DB2 376
Escenarios de instalación del Centro de información de DB2 378
Instalación del Centro de información de DB2 utilizando el asistente de instalación de DB2 (UNIX) 380
Instalación del Centro de información de DB2 utilizando el asistente de instalación de DB2 (Windows) 383
Invocación del Centro de información de DB2 . . 385
Actualización del Centro de información de DB2 instalado en el sistema o en un servidor de intranet 386
Visualización de temas en el idioma preferido en el Centro de información de DB2 387
Documentación PDF e impresa de DB2 388
Información básica de DB2 388
Información de administración 389
Información para el desarrollo de aplicaciones 389
Información de Business Intelligence 390
Información de DB2 Connect 390
Información de iniciación 391
Información de aprendizaje 391
Información sobre componentes opcionales . . 392
Notas del release 392
Impresión de manuales de DB2 desde archivos PDF 393
Solicitud de manuales de DB2 impresos 394
Invocación de ayuda según contexto desde una herramienta de DB2 394
Invocación de la ayuda de mensajes desde el procesador de línea de mandatos 396
Invocación de la ayuda de mandatos desde el procesador de línea de mandatos 396
Invocación de la ayuda para estados de SQL desde el procesador de línea de mandatos 397
Guías de aprendizaje de DB2 397
Información de resolución de problemas de DB2 398
Accesibilidad 399
Entrada de teclado y navegación 399
Pantalla accesible 399
Compatibilidad con tecnologías de asistencia 400
Documentación accesible 400
Diagramas de sintaxis en formato decimal con puntos. 400
Certificación Common Criteria de productos DB2 Universal Database 402

Apéndice D. Avisos 403
Marcas registradas. 405

Índice. 407

Cómo ponerse en contacto con IBM 417
Información sobre productos 417

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. Rutinas

Capítulo 1. Introducción a las rutinas

Rutinas en el desarrollo de aplicaciones 3
Tipos de rutinas (procedimientos, funciones, métodos) 5
Rutinas definidas por el usuario 10
Comparación de procedimientos, funciones y métodos 13
Procedimientos 13
Funciones escalares definidas por el usuario . . 15
Funciones escalares definidas por el usuario . . 17
Métodos 18

Rutinas en el desarrollo de aplicaciones

Una rutina es un objeto de base de datos que puede encapsular lógica de programación y bases de datos relacionada con una tarea específica. Existen tres tipos de rutinas: procedimientos, funciones y métodos. Cada tipo de rutina proporciona una interfaz diferente para contener la lógica y las operaciones de base de datos que pueden utilizarse para ampliar las funciones de una sentencia de SQL o una aplicación cliente. Debe tomar en consideración las numerosas ventajas implicadas en crear y utilizar rutinas cuando ha de desarrollar o actualizar una aplicación de base de datos.

Cuando se enfrente a la tarea de desarrollar nuevas funciones que interactúen con una base de datos, existen dos enfoques entre los cuales puede elegir. Puede añadir la nueva lógica a una aplicación cliente o puede crear una rutina, en la que la nueva lógica residirá en el servidor de bases de datos. La elección del último enfoque brinda varios beneficios.

Beneficios del uso de rutinas:

Puede obtener los beneficios siguientes si mueve la lógica de aplicaciones a las rutinas:

Encapsular la lógica de aplicaciones
En un entorno con numerosos sistemas cliente, cada uno de ellos ejecutando varias aplicaciones de base de datos, el uso eficaz de rutinas puede simplificar la reutilización, estandarización y mantenimiento del código. Por ejemplo, si es necesario cambiar un aspecto determinado del comportamiento de una aplicación en un entorno en el que se utilizan rutinas, sólo requerirá modificación la rutina afectada que encapsule ese comportamiento. Si no se utilizaran rutinas en esta instancia, se habrían requerido cambios de la lógica de aplicaciones en cada aplicación cliente.

Permitir un acceso controlado a objetos de base de datos
Puede utilizar rutinas para controlar el acceso a objetos de base de datos. Puede que un usuario no tenga permiso para emitir generalmente una sentencia de SQL determinada; sin embargo, se le puede otorgar permiso para invocar rutinas que contengan implementaciones específicas de estas sentencias.

Reducir el tráfico de la red
Cuando se ejecuta una aplicación en un sistema cliente, se envía por separado cada sentencia de SQL desde el sistema cliente al sistema servidor, y cada resultado se devuelve por separado. Esto puede derivar en un elevado tráfico de red. Si puede identificarse un trabajo que implica una gran actividad de base de datos y poca interacción del usuario, tiene sentido instalar este trabajo en el servidor. Al ejecutarse este trabajo en el

servidor, se reduce la cantidad de tráfico de red entre el sistema cliente y el sistema servidor. Las rutinas de DB2 se ejecutan en el servidor de bases de datos de esta manera. El uso de rutinas es una forma eficaz de reducir el tráfico de la red y de mejorar el rendimiento global de las aplicaciones cliente.

Aliviar la carga de proceso en el cliente
En entornos en los que el rendimiento de un sistema cliente es importante, las rutinas constituyen un medio práctico para reducir la dependencia del sistema cliente. Después de que una aplicación invoque una rutina, el proceso de la rutina se realiza en el servidor de bases de datos, lo que permite que la aplicación aproveche la potencia del servidor de bases de datos a la vez que aligera la carga de proceso del sistema cliente.

Permitir una ejecución más rápida y eficaz
Las rutinas son objetos de base de datos y, por lo tanto, tienen una relación más estrecha con el gestor de bases de datos que las aplicaciones cliente. Para algunos tipos de rutinas, el rendimiento de las sentencias de SQL puede resultar mucho mejor que el de las sentencias de SQL ejecutadas desde una aplicación cliente. Por ejemplo, las rutinas NOT FENCED se ejecutan en el mismo proceso que el gestor de bases de datos utilizando la memoria compartida para la comunicación. De este modo, las rutinas se vuelven más competentes en la transmisión de peticiones de SQL y datos de lo que podría una aplicación cliente que se comunica utilizando los protocolos TCP/IP.

Interoperatividad de las implementaciones lógicas
Puesto que los distintos programadores implementan a menudo módulos de código, cada uno con experiencia de programación en diferentes lenguajes de programación, y puesto que, generalmente, es aconsejable reutilizar el código siempre que sea posible para ahorrar en tiempo y costes de desarrollo, las rutinas de DB2® son altamente interoperativas.

- Una aplicación cliente en un lenguaje de programación puede invocar rutinas que estén implementadas en otro lenguaje de programación. Por ejemplo, las aplicaciones cliente en C pueden invocar rutinas de ejecución en el lenguaje común .NET.
- Una rutina puede invocar otra rutina independientemente del tipo de rutina o del lenguaje de implementación de la rutina. Por ejemplo, un procedimiento de Java™ (un tipo de rutina) puede invocar una función escalar de SQL (otro tipo de rutina con un lenguaje de implementación diferente).
- Una rutina creada en un servidor de bases de datos de un sistema operativo puede invocarse desde un cliente DB2 que se ejecute en otro sistema operativo.

Existen varias clases de rutinas dirigidas a determinadas necesidades funcionales, así como varias implementaciones de rutinas. La elección del tipo de rutina y su implementación puede afectar al grado en que se presentan los beneficios anteriores. En general, las rutinas son una forma efectiva de encapsular la lógica a fin de poder ampliar el SQL y mejorar la estructura, el mantenimiento y, potencialmente, el rendimiento de las aplicaciones.

Conceptos relacionados:

- "Procedimientos" en la página 13
- "Invocación de la rutina" en la página 209
- "Lenguajes de programación de rutinas soportados" en la página 21

- "Funciones escalares definidas por el usuario" en la página 15
- "Métodos" en la página 18
- "Funciones escalares definidas por el usuario" en la página 17

Tareas relacionadas:

- "Creación de rutinas JDBC" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de rutinas SQLJ" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de rutinas C para UNIX" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de rutinas C++ para UNIX" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de rutinas IBM COBOL en AIX" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de rutinas Micro Focus COBOL para UNIX" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de rutinas C/C++ en Windows" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de rutinas IBM COBOL en Windows" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de rutinas Micro Focus COBOL en Windows" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Escritura de rutinas" en la página 36
- "Creación de rutinas en la base de datos" en la página 35
- "Depuración de rutinas" en la página 42

Tipos de rutinas (procedimientos, funciones, métodos)

Las rutinas se agrupan principalmente según su funcionalidad; no obstante, también se pueden agrupar según su implementación. Los tres tipos de rutinas funcionales más importantes son: los procedimientos (también denominados procedimientos almacenados), las funciones y los métodos. Existen varias implementaciones posibles para las rutinas, incluidas las siguientes: incorporadas, con fuente, SQL y externas. Primero, se describen los tipos de rutinas funcionales y, a continuación, siguen las explicaciones de las posibles implementaciones.

Tipos de rutinas funcionales:

Procedimientos

Los procedimientos, también denominados procedimientos almacenados, sirven de extensiones de subrutina a las aplicaciones cliente, rutinas, activadores y sentencias dinámicas compuestas. Los procedimientos se invocan ejecutando la sentencia CALL con una referencia a un procedimiento.

Funciones

Una función es una relación entre un conjunto de valores de datos de entrada y un conjunto de valores de resultado. Las funciones le permiten ampliar y personalizar el SQL. Las funciones se invocan desde elementos de sentencias de SQL tales como una lista de selección o una cláusula FROM. Existen cuatro tipos de funciones: funciones de agregación, funciones escalares, funciones de fila y funciones de tabla.

Funciones de agregación

También denominada función de columna, este tipo de función devuelve un valor escalar que es el resultado de un cálculo con un conjunto de valores de entrada semejantes. Los valores de entrada similares se pueden especificar, por ejemplo, mediante una columna dentro de una tabla o mediante tuplas en una cláusula VALUES. Este conjunto de valores se conoce como conjunto de argumentos. Por ejemplo, la consulta siguiente busca la cantidad total de cerrojos de los que se dispone o los que están pedidos, incluidas todas las clases de cerrojos, utilizando la función de agregación SUM:

```
SELECT SUM(qinstock + qonorder)
FROM inventory
WHERE description LIKE '%Bolt%'
```

Una función de agregación no se puede implementar como una función externa; sólo como una función con fuente que procede de una función de agregación incorporada.

Funciones escalares

Una función escalar es una función que, para cada conjunto de uno o más parámetros escalares, devuelve un solo valor escalar. Los ejemplos de funciones escalares incluyen `length` y `substr`. También se puede crear una función escalar que efectúe un cálculo matemático complejo sobre los parámetros de entrada. Las funciones escalares se pueden referenciar en cualquier parte donde una expresión sea válida dentro de una sentencia de SQL, como, por ejemplo, en una lista de selección o en una cláusula FROM. Las funciones escalares se pueden implementar como funciones externas o funciones con fuente.

Funciones de fila

Una función de fila es una función que, para cada conjunto de uno o más parámetros escalares, devuelve una sola fila. Las funciones de fila sólo se pueden utilizar como atributos de correlación de funciones de transformación de un tipo estructurado en valores de tipos de datos incorporados de una fila. Una función de fila sólo se puede implementar como función de SQL.

Funciones de tabla

Las funciones de tabla son funciones que, para un grupo de conjuntos de uno o más parámetros, devuelven una tabla a la sentencia de SQL que hace referencia a las mismas. Sólo se puede hacer referencia a las funciones de tabla en la cláusula FROM de una sentencia SELECT. La tabla devuelta por una función de tabla puede participar en uniones, operaciones de agrupación, operaciones como UNION y cualquier operación que se pueda aplicar a una vista de sólo lectura. Las funciones de tabla se pueden implementar en SQL o en un lenguaje de programación externo.

Métodos

Una encapsulación de lógica que proporciona el comportamiento de los tipos estructurados. Un tipo estructurado es un tipo de datos definido por el usuario que contiene uno o más atributos con nombre, cada uno de los cuales tiene un tipo de datos. Los atributos son propiedades que describen

una instancia de un tipo. Por ejemplo, una figura geométrica puede tener atributos tales como su lista de coordenadas cartesianas. Un método se implementa generalmente para un tipo estructurado a fin de representar una operación sobre los atributos del tipo estructurado. Para una figura geométrica, un método puede consistir en calcular el volumen de ésta. Los métodos se pueden implementar como métodos de SQL o externos.

El diagrama siguiente ilustra la jerarquía de clasificación de las rutinas:

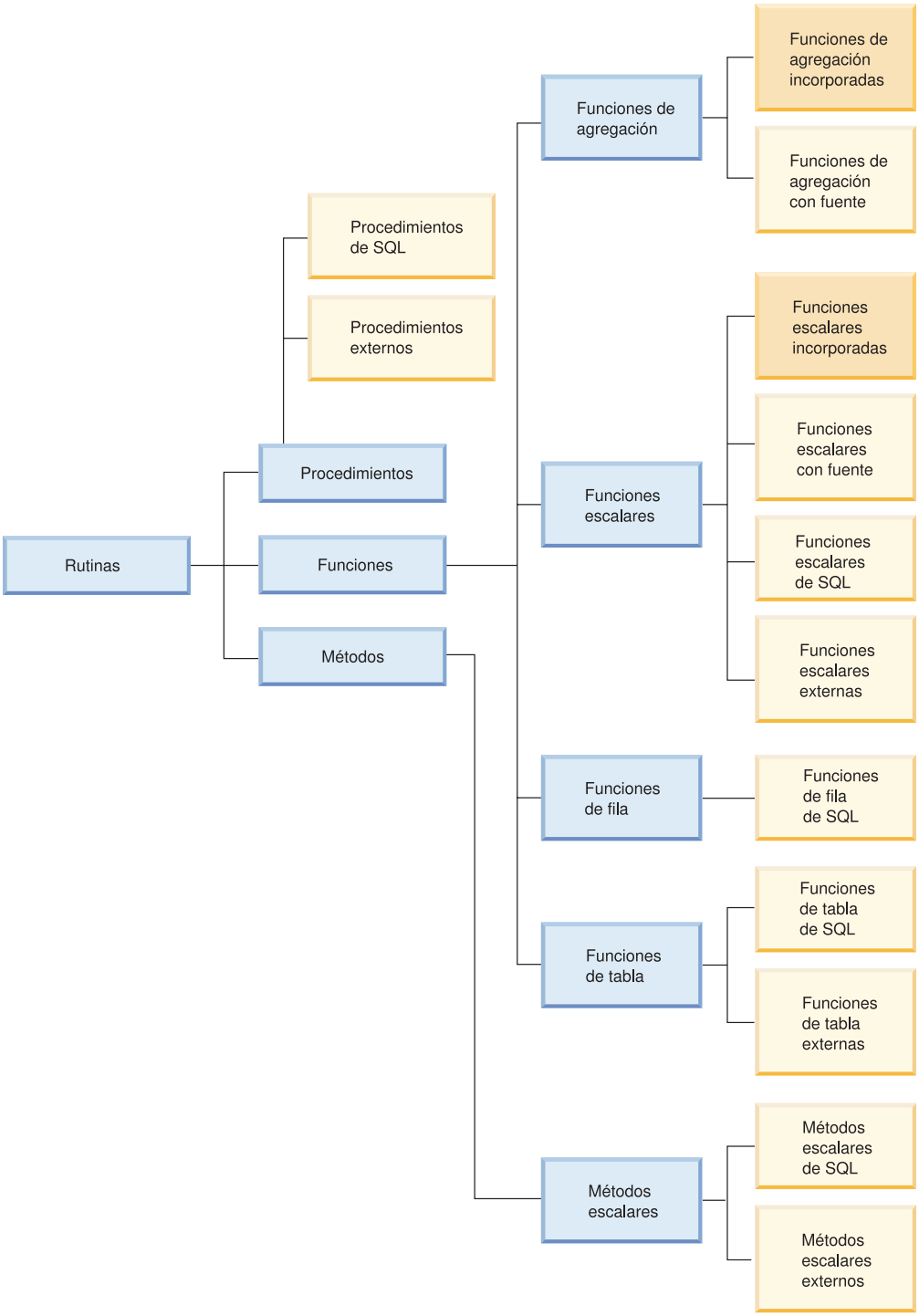


Figura 1. Clasificaciones de las rutinas

Tipos de implementaciones de rutinas:

Existen varias implementaciones posibles para las rutinas: incorporadas, con fuente, SQL y externas.

Incorporadas

Algunas rutinas están incorporadas en el código del sistema DB2. Estas rutinas tienen una tipificación estricta y funcionan bien porque su lógica es

nativa para el código de base de datos. Estas rutinas se encuentran en el esquema SYSIBM. Algunos ejemplos de funciones escalares y de agregación incorporadas incluyen:

Funciones escalares incorporadas
+, −, *, /, ||, substr, concat, length, char, decimal days

Funciones de agregación incorporadas
avg, count, min, max, stdev, sum, variance

Las funciones incorporadas comprenden la mayor parte de la funcionalidad aritmética, de manipulación de series y de conversión que normalmente resulta necesaria. Puede utilizar estas funciones de inmediato en sus sentencias de SQL. Para obtener una lista completa de las funciones incorporadas disponibles, vea la Consulta de SQL.

Las otras implementaciones están motivadas por el usuario. Estas implementaciones, a diferencia de las funciones incorporadas, requieren que el usuario cree explícitamente la rutina utilizando la sentencia CREATE adecuada para el tipo de rutina. Las funciones creadas por el usuario y los procedimientos se encuentran en el esquema SYSTOOLS.

Con fuente
Una función con fuente es una función que duplica la semántica de otra función, conocida como su función fuente. Actualmente, sólo las funciones escalares y de agregación pueden ser funciones con fuente. Las funciones con fuente son particularmente útiles para permitir que un tipo diferenciado herede, de forma selectiva, la semántica de su tipo fuente. Básicamente, las funciones con fuente son una forma especial de implementación de SQL para una función.

SQL

Una rutina de SQL se compone enteramente de sentencias de SQL. Estas sentencias se especifican en la sentencia CREATE que se utiliza para crear la rutina en la base de datos. El SQL Procedural Language (SQL PL) es una extensión de lenguaje del SQL básico que consta de sentencias y elementos de lenguaje que se pueden utilizar para implementar la lógica de programación en SQL.

El SQL PL incluye un conjunto de sentencias para declarar variables y manejadores de condiciones (sentencia DECLARE), asignar valores a las variables (sentencia de asignación) e implementar la lógica de procedimiento (sentencias de control: IF, WHILE, FOR, GOTO, LOOP, SIGNAL, etc.). El SQL que incluye el SQL PL o, si está limitado, un subconjunto del SQL PL, se puede utilizar para crear procedimientos, funciones y métodos de SQL.

Externas
Una rutina que se crea en la base de datos utilizando la sentencia CREATE específica del tipo de rutina, pero que tiene su lógica de rutina implementada en una aplicación de lenguaje principal de programación externo. La asociación de la rutina con la aplicación de código externo está indicada mediante la especificación de la cláusula EXTERNAL en la sentencia CREATE. Las rutinas externas se pueden escribir en C, C++, Java™, OLE, así como en los lenguajes de programación soportados de ejecución en el lenguaje común .NET. Los procedimientos externos también se pueden escribir en COBOL.

Rutinas proporcionadas por DB2:

Además, DB2 proporciona algunos procedimientos y funciones en los esquemas SYSPROC, SYSFUN y SYSTOOLS que se pueden utilizar. Aunque estas rutinas adicionales se suministran con DB2, no son rutinas incorporadas. En lugar de ello, se implementan como rutinas definidas por el usuario preinstaladas. Habitualmente, estas rutinas encapsulan una función de programa de utilidad. Algunos ejemplos de éstas son: `SNAPSHOT_TABLE`, `HEALTH_DB_HI`, `SNAPSHOT_FILEW`, `REBIND_ROUTINE_PACKAGE`. Puede utilizar inmediatamente estas funciones y procedimientos siempre que tenga el esquema SYSPROC y el esquema SYSFUN en CURRENT PATH. Estos esquemas están incluidos en CURRENT PATH por omisión.

Conceptos relacionados:

- "Rutinas en el desarrollo de aplicaciones" en la página 3
- "Consideraciones sobre el rendimiento para crear rutinas" en la página 24
- "Consideraciones sobre seguridad para las rutinas" en la página 27

Rutinas definidas por el usuario

DB2® proporciona rutinas incorporadas que capturan la funcionalidad de la mayoría de las funciones de conversión, de serie y aritméticas utilizadas normalmente; no obstante, a fin de que pueda encapsular su propia lógica, DB2 permite al usuario crear sus propias rutinas. Estas rutinas se conocen como rutinas definidas por el usuario. Puede crear sus propios procedimientos, funciones y métodos en cualquiera de los estilos de implementación soportados para el tipo de rutina. Generalmente, la expresión 'definido por el usuario' no se utiliza cuando se hace referencia a procedimientos y a métodos; sin embargo, las funciones definidas por el usuario se suelen denominar UDF.

Sentencias CREATE para las rutinas:

Los procedimientos, funciones y métodos definidos por el usuario se crean en la base de datos ejecutando la sentencia CREATE adecuada para la clase de rutina. Estas sentencias de creación de rutinas incluyen: CREATE PROCEDURE, CREATE FUNCTION y CREATE METHOD. Las cláusulas específicas de cada una de las sentencias CREATE definen las características de la rutina, tales como el nombre de la rutina, el número y tipo de argumentos de la rutina y detalles sobre la lógica de la misma. DB2 utiliza la información proporcionada por las cláusulas para identificar y ejecutar la rutina cuando ésta se invoca. Una vez ejecutada satisfactoriamente la sentencia CREATE para una rutina, ésta se crea en la base de datos. Las características de la rutina se almacenan en las tablas del sistema DB2 que los usuarios pueden consultar. La ejecución de la sentencia CREATE para crear una rutina también se conoce como definición de una rutina o registro de una rutina.

Implementación de la lógica de una rutina:

Existen tres estilos de implementación que se pueden utilizar para especificar la lógica de una rutina: SQL, externa y con fuente. Estas implementaciones se comparan a continuación para que pueda ver las ventajas y utilidad de cada implementación:

SQL

La lógica de una rutina de SQL se escribe enteramente en SQL, que se especifica dentro del cuerpo de la sentencia CREATE de la rutina. El cuerpo de un procedimiento, función o método de SQL puede estar compuesto por sentencias de SQL y sentencias de SQL PL.

Las rutinas de SQL son rápidas y fáciles de implementar a causa de su sintaxis sencilla y funcionan bien debido a su estrecha relación con DB2 para las implementaciones que contienen, en su mayor parte, sentencias de SQL y usos menos complejos de la lógica de SQL PL. En el caso de los procedimientos de SQL, se proporciona, asimismo, soporte para un manejo de errores fácil de utilizar. No obstante, las rutinas de SQL están limitadas en que no pueden efectuar directamente llamadas al sistema ni tampoco operaciones con entidades que residan fuera de la base de datos, y, según el tipo funcional de la rutina, puede que no den soporte a la ejecución de todas las sentencias de SQL.

Externas

Las rutinas externas son rutinas cuya lógica está implementada en una biblioteca creada por el usuario o clase que reside en el sistema de archivos del servidor de bases de datos - externa a la propia base de datos. La biblioteca o clase se puede compilar a partir de una aplicación fuente escrita en uno de los lenguajes principales de programación siguientes: C, C++, Java™, OLE o cualquier lenguaje compatible con .NET. Los procedimientos externos también se pueden escribir en COBOL. Todas las rutinas externas pueden contener SQL, excepto las rutinas de OLE.

Generalmente, las rutinas externas son algo más complejas en su implementación que las rutinas de SQL; no obstante, resultan muy potentes porque, con una rutina externa, se puede aprovechar la total funcionalidad y rendimiento del lenguaje de programación elegido para la implementación. Además, las funciones externas tienen la ventaja de permitir el acceso y la manipulación de entidades que residen fuera de la base de datos, tales como una red o un sistema de archivos. Para las rutinas que requieran un grado menor de interacción con la base de datos DB2, pero que deban contener mucha lógica o bien lógica más compleja, es preferible la implementación de una rutina externa. Por ejemplo, el uso de las rutinas externas es ideal para implementar nuevas funciones que realicen operaciones con los tipos de datos incorporados y amplíen su utilidad, como, por ejemplo, una nueva función de serie que realice operaciones con un tipo de datos VARCHAR o una función matemática complicada que realice operaciones con un tipo de datos DOUBLE. También son ideales para la lógica que puede representar una acción externa, como enviar un mensaje de correo electrónico. Si no encuentra problemas en programar en uno de los lenguajes de programación soportados y tiene que encapsular lógica con más hincapié en la lógica de programación que en los accesos a bases de datos, una vez que aprenda los pasos simples implicados en la creación de estas rutinas, pronto descubrirá su potencia.

Con fuente

El estilo de implementación con fuente sólo es aplicable a las funciones. La lógica de una función con fuente deriva de una función fuente existente. La función fuente se identifica simplemente especificando la cláusula SOURCE en la sentencia `CREATE FUNCTION` especial (con fuente o plantilla). No existe un lenguaje particularmente asociado con esta implementación. El uso más habitual de las funciones con fuente consiste en permitir que un tipo diferenciado herede, de forma selectiva, algunos de los operadores

y funciones que se aplican a su tipo fuente. La implementación de las funciones con fuente es sencilla, y resultan útiles si desea renombrar una función existente.

Visión general del desarrollo de las rutinas definidas por el usuario:

Para obtener ayuda en el desarrollo de rutinas, puede utilizar el Centro de Desarrollo de DB2. Éste proporciona interfaces sencillas y un conjunto de asistentes que facilitan la realización de tareas de desarrollo. También puede integrar el Centro de Desarrollo de DB2 con herramientas populares para el desarrollo de aplicaciones, como por ejemplo Microsoft® Visual Studio. Como alternativa, también puede desarrollar rutinas definidas por el usuario mediante el procesador de línea de mandatos de DB2.

El desarrollo de las rutinas definidas por el usuario implica las tareas siguientes:

1. Crear la rutina en la base de datos. Esta tarea, también conocida como definición o registro de una rutina, se puede realizar en cualquier momento antes de invocar la rutina, excepto en las circunstancias siguientes:
   - Para las rutinas de Java que hacen referencia a un archivo o archivos JAR externos, el código y los archivos JAR externos se deben codificar y compilar antes de que la rutina se cree en la base de datos utilizando la sentencia CREATE específica del tipo de rutina.
   - Las rutinas que ejecutan sentencias de SQL y se refieren directamente a sí mismas se deben crear en la base de datos emitiendo la sentencia CREATE antes de que se precompile y enlace el código externo asociado con la rutina. Esto también es aplicable a las situaciones en que exista un ciclo de referencias, como por ejemplo: La Rutina A hace referencia a la Rutina B, la cual hace referencia a la Rutina A.
2. Para las rutinas externas, codificar la lógica de la rutina. La lógica de las rutinas de SQL está incluida en la sentencia CREATE de la rutina de SQL.
3. Para las rutinas externas, crear (precompilar -- en el caso de las rutinas con SQL intercalado, compilar y enlazar) la rutina. (Consulte los enlaces relacionados para obtener información, específica del sistema operativo y lenguaje, acerca de la creación.)
4. Depurar y probar la rutina.
5. Otorgar el privilegio EXECUTE sobre la rutina al invocador o invocadores de la rutina.
6. Invocar la rutina.

Conceptos relacionados:

- “Procedimientos” en la página 13
- “SQL en rutinas externas” en la página 110
- “Tipos de rutinas (procedimientos, funciones, métodos)” en la página 5
- “Funciones de tabla definidas por el usuario” en la página 63
- “Funciones escalares definidas por el usuario” en la página 15
- “Métodos” en la página 18
- “Niveles de acceso de SQL en rutinas de SQL” en la página 69
- “SQL Procedural Language (SQL PL) en DB2” en la página 67

Comparación de procedimientos, funciones y métodos

Existen tres tipos de rutinas que puede desarrollar: procedimientos, funciones definidas por el usuario (UDF) y métodos. Mientras que los detalles implicados en su creación e implementación son similares, cada uno de ellos cumple objetivos distintos.

En los apartados siguientes se presentan las características de cada tipo de rutina en un formato que facilita la comparación entre ellos. Observe que existen dos apartados dedicados a las UDF: funciones escalares definidas por el usuario y funciones para tablas definidas por el usuario. Se diferencian suficientemente como para justificar una atención individual.

Procedimientos

Un procedimiento, también denominado procedimiento almacenado, es un objeto de base de datos creado ejecutando la sentencia CREATE PROCEDURE que puede encapsular lógica y sentencias de SQL. Los procedimientos se utilizan como extensiones de subrutina para las aplicaciones, así como otros objetos de base de datos que pueden contener lógica.

Características

- Permite la encapsulación de sentencias de SQL, invocaciones de función y elementos de lógica que formulan un modelo de subrutina determinado que se puede reutilizar.
- Los procedimientos se pueden llamar desde aplicaciones cliente, otras rutinas, activadores y sentencias dinámicas compuestas. Los procedimientos se llaman utilizando la sentencia CALL.
- Los procedimientos pueden devolver varios conjuntos de resultados.
- Los procedimientos pueden contener sentencias de SQL que lean o modifiquen datos de tabla en bases de datos de una sola partición y de varias.
- Cuando se invoca un procedimiento, el SQL y la lógica que se incluyen en él se ejecutan en el servidor. Sólo se transfieren los datos entre el cliente y el servidor de bases de datos en la llamada y en la devolución del procedimiento. Si dispone de una serie de sentencias de SQL a ejecutar dentro de una aplicación cliente y la aplicación no necesita ningún proceso entre las sentencias, esta serie de sentencias se beneficiaría de estar incluida en un procedimiento.

  Nota: Si únicamente se invoca una sentencia de SQL en un procedimiento, la actividad general para configurar esta invocación supera al beneficio obtenido con la reducción de tráfico de la red.

Limitaciones

- Los procedimientos no están pensados para que se llamen desde los elementos de una consulta de SQL. Los procedimientos sólo se pueden invocar utilizando la sentencia CALL donde esté soportada. Se pueden utilizar las funciones para expresar la lógica que transforma los valores de columna. Aunque los procedimientos pueden devolver conjuntos de resultados, se pueden utilizar funciones de tabla para devolver una tabla dentro de la cláusula FROM de una consulta de SQL.
- Otra sentencia de SQL no puede utilizar directamente los argumentos de salida de las llamadas de procedimiento.

- Los procedimientos no pueden conservar el estado entre las invocaciones.

Usos frecuentes

- Implementar las subrutinas de aplicación que específicamente encapsulan la lógica de base de datos asociada con una tarea determinada. Por ejemplo, una aplicación empresarial de gestión de información de los empleados podría utilizar un procedimiento para encapsular las operaciones de base de datos implicadas en el contrato de un empleado.

  Dicho procedimiento podría insertar información del empleado en una tabla de empleados, una tabla de departamentos y una tabla de beneficios, calcular la paga semanal del empleado basada en un parámetro de entrada y devolver el valor de la paga semanal como uno de los parámetros de salida. Otro procedimiento podría incluir un análisis estadístico de los datos de la tabla de empleados y devolver los conjuntos de resultados del análisis. Este uso de los procedimientos identifica de manera eficaz las tareas de base de datos con respecto a las tareas que no son de base de datos dentro de una aplicación.
- Estandarizar la lógica de la aplicación. Si varias aplicaciones deben realizar el acceso o modificación de la base de datos de manera similar, un procedimiento puede proporcionar una única interfaz para ese acceso o modificación. El procedimiento está disponible para su uso por parte de todas las aplicaciones. Si hay que cambiar la interfaz para acomodar un cambio en la lógica empresarial, sólo se deberá modificar el procedimiento.

Lenguajes soportados

- SQL
- C/C++
- Java™
- OLE
- COBOL
- Lenguajes de ejecución en el lenguaje común .NET

Nota: Los procedimientos de SQL están soportados de forma nativa y no requieren la instalación de un compilador.

Conceptos relacionados:

- “Rutinas en el desarrollo de aplicaciones” en la página 3
- “Modalidades de parámetros de procedimiento” en la página 46
- “Conjuntos de resultados de procedimiento” en la página 47
- “SQL Procedural Language (SQL PL) en DB2” en la página 67

Tareas relacionadas:

- “Configuración del entorno de desarrollo de aplicaciones” en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- “Creación de procedimientos de SQL desde la línea de mandatos” en la página 72
- “Llamada a procedimientos desde activadores o rutinas de SQL” en la página 219
- “Llamada a procedimientos desde aplicaciones o rutinas externas” en la página 217

Información relacionada:

- "Sentencia CALL" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE PROCEDURE" en la publicación *Consulta de SQL, Volumen 2*
- "Software de desarrollo soportado por DB2" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Funciones escalares definidas por el usuario

Las funciones definidas por el usuario (UDF) escalares le permiten ampliar y personalizar las sentencias de SQL. Se pueden invocar del mismo modo que funciones incorporadas de DB2® como, por ejemplo, LENGTH y COUNT. Es decir, se puede hacer referencia a ellas en sentencias de SQL siempre que la expresión sea válida. Las UDF escalares aceptan cero o más valores con tipo como argumentos de entrada y devuelven un solo valor después de cada invocación.

Funciones escalares de SQL definidas por el usuario:

Las UDF escalares de SQL le permiten encapsular sentencias de SQL, funciones incorporadas y otras referencias a rutinas, así como un subconjunto de sentencias de SQL PL que se pueden utilizar para implementar lógica básica de bases de datos. Las funciones escalares de SQL pueden leer y modificar datos de SQL. Las funciones de SQL ofrecen el mejor rendimiento cuando utilizan funciones incorporadas y no contienen lógica extremadamente compleja. Para una lógica extremadamente compleja, tome en consideración implementar una UDF escalar externa.

Funciones escalares externas definidas por el usuario:

Las UDF escalares externas tienen implementada su lógica en un lenguaje de programación externo. La lógica de la función puede acceder al sistema de archivos, realizar llamadas al sistema o acceder a una red. La ejecución de la lógica de la rutina de UDF escalar externa, al igual que sucede con las UDF escalares de SQL, tiene lugar en el servidor. Las UDF escalares externas pueden leer datos de SQL, pero no pueden modificar datos de SQL. Una UDF escalar externa se puede invocar repetidamente para una sola referencia de la función y puede mantener el estado entre estas invocaciones mediante un área reutilizable, que es un almacenamiento intermedio de memoria. Esto puede resultar potente si una función requiere una lógica de configuración inicial, pero costosa. La lógica de configuración se puede realizar en una primera invocación que puede emplear el área reutilizable para almacenar algunos valores cuyo acceso o actualización es posible en invocaciones subsiguientes de la función escalar.

Características de las UDF escalares de SQL y externas

- Se puede hacer referencia a ellas como parte de una sentencia de SQL en cualquier parte en que esté soportada una expresión.
- La sentencia de SQL que realiza la invocación puede utilizar directamente la salida de una UDF escalar.
- Para las funciones escalares externas definidas por el usuario, se puede mantener el estado entre las invocaciones iterativas de la función empleando un área reutilizable.
- Pueden proporcionar una ventaja para el rendimiento cuando se utilizan en predicados, puesto que se ejecutan en el servidor. Si una función se puede aplicar a una fila candidata en el servidor, puede, a menudo,

dejar de tomar en consideración la fila antes de transmitirla a la máquina cliente, con lo que se reduce la cantidad de datos que se deben pasar desde el servidor al cliente.

- Una forma excelente de crear funciones escalares a partir de funciones incorporadas existentes. Por ejemplo, puede crear una fórmula matemática compleja reutilizando las funciones escalares incorporadas junto con otra lógica.

Limitaciones

- No se puede realizar la gestión de las transacciones dentro de una UDF escalar. Es decir, no puede emitir COMMIT ni ROLLBACK dentro de una UDF escalar.
- No pueden devolver conjuntos de resultados.
- Las UDF escalares están pensadas para devolver un solo valor escalar por cada conjunto de valores de entrada.
- Las UDF escalares externas no están pensadas para que se utilicen en una sola invocación. Están diseñadas de forma que, para una sola referencia a la UDF y un conjunto determinado de valores de entrada, la UDF se invoque una vez por valor de entrada y devuelva un solo valor escalar. En la primera invocación, las UDF escalares pueden estar diseñadas para realizar algún trabajo de configuración o para almacenar información a la que se pueda acceder en invocaciones subsiguientes. Las UDF escalares de SQL se ajustan mejor a la funcionalidad que requiere una sola invocación.
- En una base de datos de una sola partición, las UDF escalares externas pueden incluir sentencias de SQL. Estas sentencias pueden leer datos de tablas, pero no pueden modificarlos. Si la base de datos tiene más de una partición, no debe haber sentencias de SQL en una UDF escalar externa.

  En series y en bases de datos particionadas, las UDF escalares de SQL pueden contener sentencias de SQL que lean datos de tablas de base de datos.

Usos frecuentes

- Ampliar el conjunto de funciones incorporadas de DB2.
- Realizar operaciones lógicas dentro de una sentencia de SQL que SQL no puede realizar de forma nativa.
- Encapsular una consulta escalar que se reutilice habitualmente como subconsulta en las sentencias de SQL. Por ejemplo, dado un código postal, buscar en una tabla la ciudad en la que se encuentra el código postal.

Lenguajes soportados

- SQL
- C/C++
- Java™
- OLE
- Lenguajes de ejecución en el lenguaje común .NET

Notas:

1. Existe una capacidad limitada para crear funciones de agregación. Conocidas también como funciones de columna, estas funciones reciben un conjunto de valores semejantes (una columna de datos) y devuelven una sola respuesta. Una función de agregación definida por el usuario sólo se puede crear si su

fuente es una función de agregación incorporada. Por ejemplo, si existe un tipo diferenciado SHOESIZE que está definido con el tipo base INTEGER, es posible definir una UDF, AVG(SHOESIZE), como función de agregación basada en la función de agregación incorporada existente AVG(INTEGER).

2. También puede crear UDF que devuelvan una fila. Éstas se conocen como UDF para filas y sólo se pueden utilizar como funciones de transformación para los tipos estructurados. La salida de una UDF para filas es una sola fila.

Conceptos relacionados:

- "Rutinas en el desarrollo de aplicaciones" en la página 3
- "Áreas reutilizables para UDF y métodos" en la página 58

Tareas relacionadas:

- "Invocación de funciones escalares o métodos" en la página 229

Información relacionada:

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

Funciones escalares definidas por el usuario

Al igual que las UDF escalares, una UDF para tablas le permite ampliar y personalizar el SQL, pero con el objetivo de generar una tabla. A las UDF para tablas sólo se las puede invocar en la cláusula FROM de una sentencia de SQL. Las UDF para tablas aceptan cero o más valores con tipo como argumentos de entrada y devuelven una tabla.

Las funciones para tablas son potentes porque le permiten hacer que casi todas las fuentes de datos aparezcan como una tabla de DB2®. Una función para tablas se puede crear fácilmente escribiendo un programa que recoja los datos deseados, los filtre conforme a algunos parámetros de entrada si conviene y los devuelva a DB2, una fila cada vez.

Características

- Se puede hacer referencia a ellas como parte de la cláusula FROM de una sentencia de SQL.
- Las funciones para tablas externas pueden efectuar llamadas al sistema operativo, leer datos de archivos o incluso acceder a datos a través de una red en una base de datos de una sola partición.
- Los resultados se pueden procesar directamente mediante la sentencia de SQL que haga referencia a la función para tablas.
- Las funciones para tablas de SQL pueden encapsular sentencias de SQL que modifiquen datos de tablas de SQL. (Sólo las funciones para tablas de SQL tienen esta propiedad.)
- Para una sola referencia de función para tablas, una función para tablas se puede invocar varias veces y puede mantener el estado entre las invocaciones mediante un área reutilizable.
- Proporcionan un conjunto de datos para su proceso.

Limitaciones

- No pueden realizar la gestión de transacciones. Esto significa que no se pueden ejecutar las sentencias COMMIT ni ROLLBACK desde una función para tablas.
- No pueden devolver conjuntos de resultados.

- No están diseñadas para invocaciones simples.
- Sólo se pueden utilizar en una cláusula FROM.
- Las funciones para tablas externas pueden leer datos de SQL, pero no pueden modificarlos. Es posible emplear las funciones para tablas de SQL de modo que incluyan sentencias que modifiquen datos de SQL.

Usos frecuentes

- Encapsular una subconsulta compleja, pero utilizada habitualmente.
- Proporcionar una interfaz tabular para datos no relacionales. Por ejemplo, leer una hoja de cálculo y generar una tabla, que luego podrá insertarse en una tabla de DB2®.

Lenguajes soportados

- SQL
- C/C++
- Java™
- OLE
- OLE DB
- Lenguajes de ejecución en el lenguaje común .NET

Conceptos relacionados:

- "Rutinas en el desarrollo de aplicaciones" en la página 3
- "Áreas reutilizables para UDF y métodos" en la página 58
- "Modelo de proceso de las funciones de tabla" en la página 63

Información relacionada:

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

Métodos

Los métodos permiten definir comportamientos de los tipos estructurados. Son como las UDF escalares, pero sólo se pueden definir para tipos estructurados. Los métodos comparten todas las características de las UDF escalares, además de las características que se indican a continuación:

Características

- Se asocian firmemente con el tipo estructurado.
- Pueden ser sensibles al tipo dinámico del tipo sujeto.

Limitaciones

- Sólo pueden devolver un valor escalar.
- Sólo se pueden utilizar con tipos estructurados.
- No se les puede invocar para tablas escritas.

Usos frecuentes

- Proporcionar operaciones sobre tipos estructurados.
- Encapsular el tipo estructurado.

Lenguajes soportados

- SQL
- C/C++
- Java™

- OLE

Conceptos relacionados:

- "Rutinas en el desarrollo de aplicaciones" en la página 3
- "Áreas reutilizables para UDF y métodos" en la página 58

Tareas relacionadas:

- "Definición del comportamiento de los tipos estructurados" en la página 271

Información relacionada:

- "Sentencia CREATE TYPE (Estructurado)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE METHOD" en la publicación *Consulta de SQL, Volumen 2*

Capítulo 2. Desarrollo de las rutinas

Lenguajes de programación de rutinas soportados 21
Prácticas óptimas para el desarrollo de rutinas . . 24
Consideraciones sobre el rendimiento para crear rutinas 24
Consideraciones sobre seguridad para las rutinas 27
Consideraciones sobre la gestión de bibliotecas y clases 30
Restricciones del uso de rutinas 32
Creación de rutinas en la base de datos 35
Escritura de rutinas. 36
Autorizaciones y enlace de rutinas que contienen SQL 38
Depuración de rutinas. 42
Conflictos de datos cuando los procedimientos leen o graban en tablas 44
Características de los procedimientos. 46
Modalidades de parámetros de procedimiento. . 46
Conjuntos de resultados de procedimiento . . . 47
Conjuntos de resultados de procedimiento . . 47
Devolución de conjuntos de resultados desde procedimientos de SQL y SQL incorporado. . 49
Devolución de conjuntos de resultados desde procedimientos de SQLJ 51
Devolución de conjuntos de resultados desde procedimientos JDBC 52
Recepción de conjuntos de resultados de procedimiento en las rutinas de SQL 53
Recepción de conjuntos de resultados de procedimiento en aplicaciones y rutinas SQLJ . 54
Recepción de conjuntos de resultados de procedimiento en aplicaciones y rutinas JDBC . 55
Manejo de los parámetros en los procedimientos de PROGRAM TYPE MAIN o PROGRAM TYPE SUB 56
Características de las UDF y los métodos 58
Áreas reutilizables para UDF y métodos. . . . 58
Áreas reutilizables en sistemas operativos de 32 bits y 64 bits 61
Modelo de proceso de los métodos y las funciones escalares 62
Funciones de tabla definidas por el usuario. . . . 63
Funciones de tabla definidas por el usuario. . . 63
Modelo de proceso de las funciones de tabla . . 63
Modelo de ejecución de funciones de tabla para Java 65

Lenguajes de programación de rutinas soportados

En general, se utilizan rutinas para mejorar el rendimiento global del sistema de gestión de bases de datos, permitiendo que las funciones de las aplicaciones se realicen en el servidor de bases de datos. La proporción de mejora obtenida con estos esfuerzos está limitada, en cierta medida, por el lenguaje elegido para escribir una rutina.

Algunos de los aspectos que debe tener en cuenta antes de implantar rutinas en un lenguaje determinado son:

- Las técnicas disponibles para desarrollar una rutina en un entorno y un lenguaje concretos.
- La fiabilidad y seguridad del código implantado por un lenguaje.
- La escalabilidad de las rutinas escritas en un lenguaje determinado.

Para ayudarle en la valoración de los criterios anteriores, a continuación se muestran algunas características de diversos lenguajes soportados:

SQL

- Las rutinas SQL son más rápidas que las rutinas Java™ y tienen un rendimiento aproximadamente equivalente al de las rutinas C/C++ NOT FENCED.
- Las rutinas SQL se escriben por completo en SQL y pueden incluir elementos en SQL Procedural Language (SQL PL), un lenguaje de alto nivel y fácil de utilizar que permite implantarlas rápidamente.
- DB2® considera que las rutinas SQL son 'seguras' porque constan enteramente de sentencias SQL. Las rutinas SQL se ejecutan siempre

directamente en el mecanismo de la base de datos, lo cual les otorga un alto nivel de rendimiento y escalabilidad.

C/C++

- Las rutinas CLI de DB2 y las de SQL incorporado en C/C++ son más rápidas que las rutinas Java. Tienen un rendimiento aproximadamente equivalente al de las rutinas SQL si se ejecutan en modalidad NOT FENCED.
- Las rutinas C/C++ tienen tendencia al error. Es recomendable registrar las rutinas C/C++ como FENCED NOT THREADSAFE, puesto que las rutinas en estos lenguajes son las que más tienden a interrumpir el funcionamiento del mecanismo de la base de datos de DB2 al causar una corrupción de la memoria. La ejecución en modalidad FENCED NOT THREADSAFE, mientras que es más segura, provoca una actividad general de rendimiento.

  A fin de informarse acerca de la manera de valorar y mitigar los riesgos de registrar las rutinas C/C++ como NOT FENCED o FENCED THREADSAFE, consulte el tema ″Consideraciones sobre seguridad para las rutinas″.
- Por omisión, las rutinas C/C++ se ejecutan en modalidad FENCED NOT THREADSAFE a fin de aislarlas de posibles daños en la ejecución de otras rutinas. Como consecuencia, tendrá un proceso `db2fmp` por rutina C/C++ que se ejecute simultáneamente en el servidor de bases de datos. Esto puede generar problemas de escalabilidad en algunos sistemas.

Java

- Las rutinas Java son más lentas que las rutinas C/C++ o SQL.
- Las rutinas Java son más seguras que las rutinas C/C++ porque la JVM maneja el control de las operaciones peligrosas. A causa de esto, se incrementa la fiabilidad, ya que resulta difícil que una rutina Java dañe a otra rutina que se ejecuta en el mismo proceso.

  Nota: Para evitar operaciones potencialmente peligrosas, no se permiten llamadas JNI (Java Native Interface) desde las rutinas Java. Si es necesario invocar código C/C++ desde una rutina Java, podrá conseguirlo invocando una rutina C/C++ catalogada separadamente.
- Si se ejecutan en modalidad FENCED THREADSAFE (valor por omisión), las rutinas Java se escalan bien. Todas las rutinas Java FENCED compartirán unas cuantas JVM (si la pila de Java para un proceso db2fmp determinado está cercana a agotarse, en el sistema puede haber más de una JVM en uso).
- Actualmente las rutinas Java NOT FENCED no reciben soporte. Una rutina Java definida como NOT FENCED se invocará como si se hubiera definido como FENCED THREADSAFE.

Lenguajes de ejecución en el lenguaje común .NET

- Las rutinas de ejecución en el lenguaje común (CLR) .NET son rutinas que se compilan en el código de bytes del lenguaje intermedio (IL) que se puede interpretar mediante la CLR de .NET Framework. El código fuente de una rutina CLR puede estar escrito en cualquier lenguaje soportado por .NET Framework.
- Trabajar con las rutinas .NET CLR permite al usuario la flexibilidad de codificar en el lenguaje de programación .NET CLR soportado que elija.

- Los ensamblajes CLR se pueden crear a partir de subensamblajes que se han compilado con diferente código fuente de lenguaje de programación .NET, lo que permite al usuario reutilizar e integrar módulos de código escritos en varios lenguajes.
- Las rutinas CLR sólo se pueden crear como rutinas FENCED NOT THREADSAFE. Esto minimiza la posibilidad de corrupción en el mecanismo, pero también significa que estas rutinas no se pueden beneficiar de la oportunidad que ofrecen las rutinas NOT FENCED en cuanto al rendimiento.

OLE

- Las rutinas de OLE se pueden implantar en Visual C++, en Visual Basic y en otros lenguajes soportados por OLE.
- La velocidad de las rutinas automatizadas OLE dependerá del lenguaje utilizado para implantarlas. En general, son más lentas que las rutinas C/C++ que no son OLE.
- Las rutinas de OLE sólo se pueden ejecutar en modalidad FENCED NOT THREADSAFE. Así se minimiza la posibilidad de corrupción en el mecanismo. Esto también significa que las rutinas automatizadas OLE no se escalan bien.

OLE DB

- OLE DB sólo se puede utilizar para definir funciones de tabla.
- Las funciones de tabla OLE DB se conectan a una fuente de datos OLE DB externa.
- Dependiendo del proveedor de OLE DB, las funciones de tabla OLE DB son generalmente más rápidas que las funciones de tabla Java, pero más lentas que las funciones de tabla incorporadas al SQL o C/C++. No obstante, algunos predicados de la consulta en que se invoca la función se pueden evaluar en el proveedor de OLE DB, por lo que quedará reducido el número de filas que DB2 tenga que procesar. Con frecuencia, esto produce una mejora en el rendimiento.
- Las rutinas de OLE DB sólo se pueden ejecutar en modalidad FENCED NOT THREADSAFE. Así se minimiza la posibilidad de corrupción en el mecanismo. Esto también significa que las funciones de tabla automatizadas OLE DB no se escalan bien.

Conceptos relacionados:

- "Consideraciones sobre el rendimiento para crear rutinas" en la página 24
- "Consideraciones sobre seguridad para las rutinas" en la página 27
- "Rutinas C/C++" en la página 164
- "Rutinas Java" en la página 181
- "SQL Procedural Language (SQL PL) en DB2" en la página 67

Tareas relacionadas:

- "Creación de rutinas JDBC" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de rutinas SQLJ" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de procedimientos SQL" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Building CLI routines on UNIX" en la publicación *CLI Guide and Reference, Volume 1*

- "Creación de rutinas C para UNIX" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de rutinas C++ para UNIX" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Building CLI routines on Windows" en la publicación *CLI Guide and Reference, Volume 1*
- "Creación de rutinas C/C++ en Windows" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Prácticas óptimas para el desarrollo de rutinas

Los apartados que siguen contienen prácticas recomendadas para el desarrollo de rutinas seguras que funcionen bien.

Consideraciones sobre el rendimiento para crear rutinas

Uno de los beneficios más significativos de crear rutinas, en lugar de expandir aplicaciones cliente, es el rendimiento. Tome en consideración los siguientes efectos en el rendimiento a la hora de elegir un enfoque para la implementación de rutinas.

Modalidad NOT FENCED

Una rutina NOT FENCED se ejecuta en el mismo proceso que el gestor de bases de datos. En general, la ejecución de la rutina como NOT FENCED genera un mejor rendimiento en comparación con su ejecución en modalidad FENCED, puesto que las rutinas FENCED se ejecutan en un proceso especial de DB2®, fuera del espacio de direcciones del mecanismo.

Mientras que son de esperar mejoras en el rendimiento de las rutinas durante la ejecución de las mismas en modalidad NOT FENCED, el código del usuario puede corromper accidental o intencionadamente la base de datos o dañar las estructuras de control de ésta. Sólo debe utilizar rutinas NOT FENCED cuando tenga necesidad de maximizar los beneficios en el rendimiento, y si considera que la rutina es segura. (A fin de informarse acerca de la manera de valorar y mitigar los riesgos de registrar las rutinas C/C++ como NOT FENCED, consulte el tema ″Consideraciones sobre seguridad para las rutinas″.) Si la rutina no es suficientemente segura como para ejecutarla en el proceso del gestor de bases de datos, utilice la cláusula FENCED cuando la registre. Para limitar la creación y ejecución de código potencialmente inseguro, DB2 requiere que un usuario tenga un privilegio especial, CREATE_NOT_FENCED_ROUTINE, a fin de crear rutinas NOT FENCED.

Si se produce una terminación anómala mientras se ejecuta una rutina NOT FENCED, el gestor de bases de datos intentará una recuperación adecuada si la rutina se ha registrado como NO SQL. Sin embargo, el gestor de bases de datos fallará para las rutinas no definidas como NO SQL.

Las rutinas NOT FENCED se deben precompilar con la opción WCHARTYPE NOCONVERT si utilizan datos GRAPHIC o DBCLOB.

Modalidad FENCED THREADSAFE

Las rutinas FENCED THREADSAFE se ejecutan en el mismo proceso que otras rutinas. Más concretamente, las rutinas que no son Java comparten un proceso, mientras que las rutinas Java™ comparten otro proceso, separado de las rutinas escritas en otros lenguajes. Esta separación protege

a las rutinas Java de las rutinas que potencialmente tienen más tendencia al error escritas en otros lenguajes. Además, el proceso para las rutinas Java contiene una JVM, que provoca un elevado coste de memoria y no se utiliza en los otros tipos de rutinas. Varias invocaciones de rutinas FENCED THREADSAFE generan un compartimiento de recursos que causa una menor actividad general del sistema que las rutinas FENCED NOT THREADSAFE, las cuales se ejecutan, cada una de ellas, en su propio proceso dedicado.

Si piensa que una rutina es suficientemente segura como para ejecutarla en el mismo proceso que otras rutinas, utilice la cláusula THREADSAFE cuando la registre. Al igual que sucede con las rutinas NOT FENCED, encontrará información acerca de la manera de valorar y mitigar los riesgos de registrar las rutinas C/C++ como FENCED THREADSAFE en el tema ″Consideraciones sobre seguridad para las rutinas″.

Si una rutina FENCED THREADSAFE termina de forma anómala, sólo se interrumpe la hebra que está ejecutando esta rutina. Las otras rutinas del proceso siguen en ejecución. Sin embargo, la anomalía que ha ocasionado que esta hebra terminara anormalmente puede afectar de forma negativa a otras hebras de rutina del proceso y causar que queden retenidas, colgadas o con datos dañados. Después de que una hebra termine anormalmente, se deja de utilizar el proceso en nuevas invocaciones de la rutina. Una vez que todos los usuarios activos finalizan sus trabajos en este proceso, éste se interrumpe.

Cuando se registran rutinas Java, se considera que son THREADSAFE a no ser que se indique lo contrario. Por omisión, todos los otros tipos de LANGUAGE son NOT THREADSAFE. Las rutinas que utilizan LANGUAGE OLE y OLE DB no se pueden especificar como THREADSAFE.

Las rutinas NOT FENCED deben ser THREADSAFE. No es posible registrar una rutina como NOT FENCED NOT THREADSAFE (SQLCODE -104).

Los usuarios de UNIX® podrán ver sus procesos THREADSAFE en Java y C buscando `db2fmp (Java)` o `db2fmp (C)`.

Modalidad FENCED NOT THREADSAFE

Las rutinas FENCED NOT THREADSAFE se ejecutan, cada una, en su propio proceso dedicado. Si ejecuta numerosas rutinas, esto puede tener un efecto perjudicial en el rendimiento del sistema de bases de datos. Si la rutina no es suficientemente segura como para ejecutarla en el mismo proceso que otras rutinas, utilice la cláusula NOT THREADSAFE cuando la registre.

En UNIX, los procesos NOT THREADSAFE aparecen como `db2fmp (pid)` (donde pid es el ID de proceso del agente que utiliza el proceso de modalidad protegida) o como `db2fmp (idle)` para un db2fmp de tipo NOT THREADSAFE agrupado.

Rutinas Java

Si pretende ejecutar una rutina Java que requiera mucha memoria, es recomendable que la registre como FENCED NOT THREADSAFE. Para las invocaciones de rutinas Java FENCED THREADSAFE, DB2 intenta elegir un proceso Java de hebras de modalidad protegida con un almacenamiento dinámico de Java que sea lo suficientemente grande como para ejecutar la rutina. Si no es posible identificar grandes consumidores de almacenamiento dinámico en su propio proceso, puede resultar que se

produzcan errores de almacenamiento dinámico fuera de Java durante los procesos Java `db2fmp` de varias hebras. Si la rutina Java no se encuentra en esta categoría, las rutinas FENCED tendrán una mejor ejecución en modalidad de hebra segura, en que pueden compartir un número reducido de JVM.

Actualmente las rutinas Java NOT FENCED no reciben soporte. Una rutina Java definida como NOT FENCED se invocará como si se hubiera definido como FENCED THREADSAFE.

Rutinas C/C++

Normalmente, las rutinas C o C++ son más rápidas que las rutinas Java, pero tienen más tendencia a presentar errores, a la corrupción de la memoria y a detenerse por anomalía grave. Por estas razones, la posibilidad de realizar operaciones de memoria hace que las rutinas C o C++ sean candidatos de riesgo para el registro en la modalidad THREADSAFE o NOT FENCED. Estos riesgos se pueden mitigar siguiendo las prácticas de programación para rutinas seguras (consulte el tema "Consideraciones sobre seguridad para las rutinas") y probando la rutina a fondo.

Rutinas de SQL

Generalmente, las rutinas de SQL, en particular los procedimientos de SQL, también son más rápidas que las rutinas Java, y su rendimiento se suele comparar con el de las rutinas C. Las rutinas SQL siempre se ejecutan en modalidad NOT FENCED y, de esta forma, proporcionan un mayor beneficio para el rendimiento que las rutinas externas. Generalmente, las UDF que contengan una lógica compleja se ejecutarán con mayor rapidez si se escriben en C que si se escriben en SQL. Si la lógica es simple, una UDF en SQL se podrá comparar con cualquier UDF externa.

Áreas reutilizables

Un área reutilizable es un bloque de memoria que se puede asignar a UDF y métodos. El área reutilizable sólo se aplica a una referencia individual a la rutina en una sentencia de SQL. Si en una sentencia existen varias referencias a una rutina, cada referencia tiene su propia área reutilizable. Un área reutilizable permite que una UDF o un método guarden su estado entre una invocación y la siguiente.

Para las UDF y los métodos cuyas inicializaciones son complejas, puede emplear áreas reutilizables a fin de almacenar los valores necesarios durante la primera invocación que se utilizarán en todas las invocaciones futuras. La lógica de otros métodos y UDF puede requerir también que se guarden los valores intermedios entre las invocaciones.

Utilización de parámetros VARCHAR en lugar de parámetros CHAR

Puede mejorar el rendimiento de las rutinas utilizando parámetros VARCHAR en lugar de parámetros CHAR en la definición de las mismas. La utilización de tipos de datos VARCHAR en lugar de tipos de datos CHAR evita que DB2 rellene los parámetros con espacios antes de pasarlos, y reduce el período de tiempo necesario para transmitir el parámetro a través de una red.

Por ejemplo, si la aplicación cliente pasa la serie "A SHORT STRING" a una rutina que espera un parámetro CHAR(200), DB2 tiene que rellenar el parámetro con 186 espacios, terminar la serie mediante un nulo y, luego, enviar a la rutina la serie de 200 caracteres entera y el terminador nulo a través de la red.

En comparación, pasar la misma serie ″A SHORT STRING″ a una rutina que espera un parámetro VARCHAR(200) da como resultado que DB2 pase simplemente la serie de 14 caracteres y un terminador nulo a través de la red.

Conceptos relacionados:

- "Opción del precompilador WCHARTYPE en C y C++" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente*
- "Opción de precompilación WCHARTYPE CONVERT" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Consideraciones sobre seguridad para las rutinas" en la página 27
- "Rutinas C/C++" en la página 164
- "Rutinas Java" en la página 181
- "Restricciones del uso de rutinas" en la página 32
- "Consideraciones sobre la gestión de bibliotecas y clases" en la página 30
- "Mejora del rendimiento de los procedimientos de SQL" en la página 82

Información relacionada:

- "Sentencia CALL" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE FUNCTION" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE PROCEDURE" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE TYPE (Estructurado)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE METHOD" en la publicación *Consulta de SQL, Volumen 2*

Consideraciones sobre seguridad para las rutinas

La creación y el despliegue de rutinas brindan una oportunidad de mejorar en gran medida el rendimiento y la eficacia de las aplicaciones de base de datos. Sin embargo, pueden existir riesgos para la seguridad si el administrador de bases de datos no gestiona correctamente el despliegue de las rutinas. En los apartados siguientes se describen los riesgos de la seguridad y los medios de los que servirse para poder mitigar tales riesgos. Después de los riesgos de la seguridad, se le presenta un apartado sobre cómo desplegar de forma segura rutinas cuya seguridad es desconocida.

Riesgos de la seguridad:

Las rutinas NOT FENCED pueden acceder a recursos del gestor de bases de datos Las rutinas NOT FENCED se ejecutan en el mismo proceso que el gestor de bases de datos. Debido a su gran proximidad al mecanismo de base de datos, las rutinas NOT FENCED pueden corromper accidental o intencionadamente la memoria compartida del gestor de bases de datos o dañar las estructuras de control de las bases de datos. Cualquier tipo de daño causará que el gestor de bases de datos falle. Asimismo, las rutinas NOT FENCED pueden corromper las bases de datos y sus tablas.

Para asegurar la integridad del gestor de bases de datos y de sus bases de datos, deberá estudiar a fondo las rutinas que va a registrar como NOT FENCED. Estas rutinas se deben probar y depurar completamente y no pueden mostrar efectos secundarios inesperados. Durante el examen de la rutina, preste gran atención a la gestión de la memoria y al uso de las variables estáticas. El potencial mayor de corrupción radica cuando el

código no gestiona adecuadamente la memoria o cuando utiliza incorrectamente las variables estáticas. Estos problemas son frecuentes en lenguajes distintos de Java™ y los lenguajes de programación .NET.

Para registrar una rutina NOT FENCED, es necesaria la autorización CREATE_NOT_FENCED_ROUTINE. Al otorgar la autorización CREATE_NOT_FENCED_ROUTINE, tenga en cuenta que el destinatario obtendrá potencialmente un acceso no restringido al gestor de bases de datos y a todos sus recursos.

Nota: Las rutinas NOT FENCED no están soportadas en las configuraciones que cumplen con los Criterios comunes.

Las rutinas FENCED THREADSAFE pueden acceder a memoria de otras rutinas FENCED THREADSAFE

Las rutinas FENCED THREADSAFE se ejecutan como hebras dentro de un proceso compartido. Cada una de estas rutinas es capaz de leer la memoria utilizada por otras hebras de rutina del mismo proceso. Por lo tanto, es posible que una rutina de hebra recoja datos sensibles de otras rutinas del proceso de hebras. Otro riesgo inherente en el compartimiento de un solo proceso es que una hebra de rutina con una gestión errónea de la memoria puede corromper otras hebras de rutina o causar que todo el proceso de hebras se detenga por anomalía grave.

Para asegurar la integridad de las otras rutinas FENCED THREADSAFE, debe estudiar a fondo las rutinas que va a registrar como FENCED THREADSAFE. Estas rutinas se deben probar y depurar completamente y no pueden mostrar efectos secundarios inesperados. Durante el examen de la rutina, preste gran atención a la gestión de la memoria y al uso de las variables estáticas. Aquí es donde radica el potencial mayor de corrupción, especialmente en los lenguajes distintos de Java.

Para registrar una rutina FENCED THREADSAFE, es necesaria la autorización CREATE_EXTERNAL_ROUTINE. Al otorgar la autorización CREATE_EXTERNAL_ROUTINE, tenga en cuenta que el destinatario es capaz, potencialmente, de supervisar o de corromper la memoria de otras rutinas FENCED THREADSAFE.

El acceso de grabación al servidor de bases de datos por parte del propietario de procesos protegidos puede provocar una corrupción del gestor de bases de datos

El ID de usuario bajo el que se ejecutan los procesos protegidos se define mediante el mandato del sistema **db2icrt** (para crear una instancia) o **db2iupdt** (para actualizar una instancia). Este ID de usuario no debe tener acceso de grabación al directorio en el que están almacenadas las clases y las bibliotecas de rutinas (en entornos UNIX®, sqllib/function; en entornos Windows®, sqllib\function). Este ID de usuario tampoco debe tener acceso de lectura ni de grabación a ninguna base de datos, ningún sistema operativo ni ningún otro directorio ni archivo importante del servidor de bases de datos.

Si el propietario de procesos protegidos tiene acceso de grabación a varios recursos importantes del servidor de bases de datos, existirá un potencial de corrupción del sistema. Por ejemplo, un administrador de bases de datos registra una rutina recibida de una fuente desconocida como FENCED NOT THREADSAFE, pensando que se puede prevenir cualquier daño potencial aislando la rutina en su propio proceso. No obstante, el ID de usuario que posee los procesos protegidos tiene acceso de grabación al directorio sqllib/function. Los usuarios invocan a esta rutina y, sin saberlo ellos, ésta sobregraba una biblioteca de sqllib/function con una versión

alternativa de un cuerpo de rutina registrado como NOT FENCED. Esta segunda rutina tiene un acceso no restringido al gestor de bases de datos al completo, así que puede distribuir información sensible de las tablas de base de datos, corromper las bases de datos, recoger información de autentificación o hacer que el gestor de bases de datos se detenga por anomalía grave.

Asegúrese de que el ID de usuario que posee los procesos protegidos no tenga acceso de grabación a directorios ni archivos importantes del servidor de bases de datos (especialmente, sqllib/function y los directorios de datos de base de datos).

Vulnerabilidad de las clases y las bibliotecas de rutinas

Si no se controla el acceso al directorio en el que están almacenadas las clases y las bibliotecas de rutinas, puede que se supriman o sobregraben unas y otras. Como se ha explicado en el punto anterior, la sustitución de un cuerpo de rutina NOT FENCED por una rutina de daño intencionado (o mal codificada) puede comprometer gravemente la estabilidad, integridad y privacidad del servidor de bases de datos y de sus recursos.

A fin de proteger la integridad de las rutinas, deberá gestionar el acceso al directorio que contiene las clases y las bibliotecas de rutinas. Asegúrese de que el menor número posible de usuarios pueda acceder a este directorio y a sus archivos. Al asignar acceso de grabación a este directorio, tenga en cuenta que este privilegio puede proporcionar al propietario del ID de usuario un acceso no restringido al gestor de bases de datos y a todos sus recursos.

Despliegue de rutinas potencialmente inseguras:

Si adquiere una rutina de una fuente desconocida, asegúrese de saber exactamente lo que hace esta rutina antes de construirla, registrarla e invocarla. Es recomendable registrarla como FENCED y NOT THREADSAFE a menos que la haya probado a fondo y no muestre efectos secundarios inesperados.

Si tiene necesidad de desplegar una rutina que no se ajusta a los criterios para rutinas seguras, regístrela como FENCED y NOT THREADSAFE. Para asegurarse de que se mantiene la integridad de la base de datos, las rutinas FENCED y NOT THREADSAFE:

- Se ejecutan en un proceso de DB2® individual, no compartido con otras rutinas. Si terminan anormalmente, el gestor de bases de datos no se verá afectado.
- Utilizan memoria distinta de la utilizada por la base de datos. Un error inadvertido en la asignación de un valor no afectará al gestor de bases de datos.

Conceptos relacionados:

- “Rutinas en el desarrollo de aplicaciones” en la página 3
- “Consideraciones sobre el rendimiento para crear rutinas” en la página 24
- “Restricciones del uso de rutinas” en la página 32
- “Consideraciones sobre la gestión de bibliotecas y clases” en la página 30

Información relacionada:

- “Sentencia CREATE FUNCTION” en la publicación *Consulta de SQL, Volumen 2*
- “Sentencia CREATE PROCEDURE” en la publicación *Consulta de SQL, Volumen 2*
- “Sentencia GRANT (Privilegios de rutina)” en la publicación *Consulta de SQL, Volumen 2*

- "Sentencia REVOKE (Privilegios de rutina)" en la publicación *Consulta de SQL, Volumen 2*

Consideraciones sobre la gestión de bibliotecas y clases

Al desarrollar rutinas para DB2®, tiene la opción de utilizar diversos lenguajes de programación, como SQL, Java™, C, C++ y lenguajes compatibles con .NET. Si desarrolla rutinas en un lenguaje que no sea SQL, éstas se conocen como rutinas externas. El código fuente compilado de una rutina externa se conoce como cuerpo de la rutina.

Protección de los cuerpos de las rutinas

Los cuerpos de las rutinas externas residen en bibliotecas y clases almacenadas en el servidor de bases de datos. DB2 no obtiene copia de seguridad de estos archivos ni los protege de ningún modo. La sentencia CREATE utilizada para crear una rutina en la base de datos añade información de definición de rutinas a los catálogos de bases de datos, incluida la información de dónde reside la biblioteca de código externo asociada con la rutina. Esta ubicación se especifica en la cláusula EXTERNAL de la sentencia CREATE. La biblioteca de la rutina o clase especificada en la cláusula EXTERNAL no está almacenada en la base de datos, pero reside en el sistema de archivos del servidor. Es fundamental para la invocación satisfactoria de las rutinas externas que la biblioteca asociada con una rutina determinada exista en la ubicación especificada en la cláusula EXTERNAL. Es posible que la biblioteca se mueva o suprima. Si esto sucede, la rutina ya no se podrá invocar satisfactoriamente.

Para preservar la integridad de las rutinas y los clientes invocantes que dependen de la rutina, debe impedir que el cuerpo de la rutina se suprima o sustituya inadvertida o intencionadamente. Esto se puede lograr gestionando el acceso al directorio que contiene la rutina y protegiendo el propio cuerpo de la rutina.

Nota: Se considera que los cuerpos de las rutinas de SQL forman parte de la base de datos y, como tales, se copiarán con otros objetos de base de datos. No obstante, al igual que las rutinas externas, sus cuerpos son propensos a experimentar modificaciones, por lo que requieren la misma protección.

Ámbito de los cuerpos de las rutinas

Para poder utilizar rutinas en una base de datos, éstas se deben catalogar con la misma base de datos. Si hay varias bases de datos en una instancia, puede catalogar rutinas externas en una base de datos utilizando cuerpos de rutina que ya se utilicen en otra base de datos. Por lo tanto, el ámbito de los cuerpos de rutina es el de la instancia. Mientras que esto permite la posibilidad de reutilizar código, se pueden producir conflictos de nombre de biblioteca o de clase en situaciones en las que no se reutiliza el código.

Concretamente, los conflictos de nombre de biblioteca o de clase se pueden producir en situaciones como la siguiente: hay varias bases de datos en una sola instancia y las rutinas de cada base de datos utilizan sus propias bibliotecas y clases de cuerpos de rutina. Se produce un conflicto cuando el nombre de una biblioteca o de una clase utilizado por una rutina de una base de datos es idéntico al nombre de la biblioteca o clase utilizado por una rutina de otra base de datos (de la misma instancia). Esto se debe a que los cuerpos de rutina se almacenan normalmente en el directorio sqllib/function, utilizado por todas las bases de datos de una instancia.

En el caso de las rutinas que no sean Java, los conflictos de nombre de biblioteca se pueden resolver mediante los pasos siguientes:

1. Almacene las bibliotecas con cuerpos de rutina en directorios distintos para cada base de datos.
2. Catalogue las rutinas con la cláusula EXTERNAL NAME, especificando la vía de acceso completa de la biblioteca en cuestión.

En el caso de las rutinas Java, los conflictos de nombre de clase no se resuelven moviendo los archivos en cuestión a directorios diferentes, porque la variable de entorno CLASSPATH tiene efecto en toda la instancia. La primera clase que se encuentra en CLASSPATH es la que se utiliza. Por lo tanto, si tiene dos rutinas Java diferentes que hacen referencia a una clase con el mismo nombre, una de las rutinas utilizará una clase incorrecta. Existen dos soluciones posibles: renombre las clases afectadas o cree una instancia distinta para cada base de datos.

Actualización de un cuerpo de rutina

Si tiene que cambiar el cuerpo de una rutina, no vuelva a compilar y a vincular la rutina con el mismo archivo (por ejemplo, sqllib/function/foo.a) que utiliza la rutina actual mientras el gestor de bases de datos se encuentre en ejecución. Si una invocación de la rutina actual accede a una versión en antememoria del proceso de la rutina y se sustituye la biblioteca subyacente, se puede provocar que la invocación de la rutina falle. Si surge la necesidad de cambiar el cuerpo de una rutina sin detener y reiniciar DB2, lleve a cabo los pasos siguientes:

1. Cree el nuevo cuerpo de la rutina con otro nombre de biblioteca o clase.
2. Vincule el nuevo cuerpo de rutina (si contiene SQL incorporado) con la base de datos.
3. Utilice la sentencia ALTER para cambiar el nombre externo (EXTERNAL NAME) de la rutina de forma que haga referencia al cuerpo actualizado de la rutina.

Una vez que ALTER actualice las entradas de catálogo de la rutina, todas las invocaciones posteriores de la rutina actualizada apuntarán al nuevo cuerpo de la rutina.

Para actualizar las rutinas Java incorporadas en archivos JAR, debe emitir una sentencia CALL SQLJ.REFRESH_CLASSES() a fin de forzar que DB2 cargue las nuevas clases. Si no emite la sentencia CALL SQLJ.REFRESH_CLASSES() después de actualizar las clases de rutinas Java, DB2 continúa utilizando las versiones anteriores de las clases. DB2 renueva las clases cuando se produce una operación COMMIT o ROLLBACK.

Nota: Si el cuerpo de rutina que se debe actualizar lo utilizan rutinas catalogadas en varias bases de datos, las acciones recomendadas en esta sección se deberán llevar a cabo para cada base de datos afectada.

Consideraciones sobre el rendimiento de la gestión de bibliotecas

El gestor de bibliotecas de DB2 ajusta dinámicamente su antememoria de bibliotecas según la carga de trabajo. Para obtener un rendimiento óptimo, tome en consideración lo siguiente:

- Haga que el número de rutinas en las bibliotecas sea lo más reducido posible. En caso de incluir diversas rutinas en la misma biblioteca,

asegúrese de agruparlas basándose en si se invocan en la misma trama de tiempo. Tome en consideración un escenario donde, en diversas aplicaciones, una llamada al procedimiento ProcA va seguida de una llamada al procedimiento ProcB. En esta situación, puede resultar adecuado incluir ProcA y ProcB en la misma biblioteca. Con un esquema de antememoria de bibliotecas, es mejor tener muchas bibliotecas más pequeñas que unas cuantas bibliotecas grandes.

- El coste de carga de una biblioteca en el proceso C se paga una sola vez para las bibliotecas que se encuentran en uso de forma coherente por parte de las rutinas C. Después de la primera invocación de la rutina, no es necesario que todas las invocaciones posteriores, de la misma hebra del proceso, carguen la biblioteca de la rutina.

Cuerpos de rutinas en bases de datos particionadas

Cuando se utilizan rutinas externas en bases de datos particionadas, la biblioteca o clase debe estar disponible en todas las particiones de la base de datos.

En UNIX®, sqllib/function es una buena ubicación para los cuerpos de rutinas, porque el directorio sqllib está montado entre todas las particiones de la base de datos.

En Windows®, una buena solución sería crear un directorio compartido al que pudieran acceder todas las particiones y colocar las bibliotecas o clases en dicho directorio.

Conceptos relacionados:

- "Consideraciones sobre el rendimiento para crear rutinas" en la página 24
- "Consideraciones sobre seguridad para las rutinas" en la página 27
- "Restricciones del uso de rutinas" en la página 32

Información relacionada:

- "Sentencia CREATE FUNCTION" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE PROCEDURE" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE METHOD" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia ALTER FUNCTION" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia ALTER METHOD" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia ALTER PROCEDURE" en la publicación *Consulta de SQL, Volumen 2*

Restricciones del uso de rutinas

A continuación se muestran restricciones para la creación de rutinas.

- En las ediciones anteriores a la Versión 8 de DB2®, CALL no era una sentencia compilada y no era obligatoria la coincidencia de tipos de datos. Los tipos de datos con que registre una rutina deben coincidir con los tipos de datos utilizados en las rutinas. Consulte las tablas que incluyen correlaciones de tipos de SQL con tipos de datos de Java™, C, automatización de OLE y OLE DB.
- Las UDF no pueden devolver conjuntos de resultados. Todos los cursores abiertos por una UDF que contiene SQL deben estar cerrados en el momento en que se complete la llamada final.
- Las rutinas no deben crear nuevas hebras.
- No se puede emitir ninguna API de nivel de conexión desde UDF o métodos.

- Desde las rutinas no es posible realizar entrada en la pantalla y teclado ni tomar salida procedente de los mismos. Por lo tanto, no debe utilizar las corrientes estándares de E/S; por ejemplo, las llamadas a `System.out.println()` en Java, `printf()` en C/C++ o `display` en COBOL. En el modelo de proceso de DB2, las rutinas se ejecutan en el fondo y no se puede escribir en pantalla. No obstante, las rutinas pueden grabar en un archivo.

  Para las rutinas FENCED que se ejecutan en UNIX®, el directorio de destino en que se debe crear el archivo, o el propio archivo, debe tener los permisos adecuados, de forma que el propietario del archivo `sqllib/adm/.fenced` pueda crearlo o grabar en él. Si se trata de rutinas NOT FENCED, el propietario de la instancia debe tener permisos de creación, lectura y grabación para el directorio en que se abra el archivo.

  Nota: DB2 no intenta sincronizar la entrada externa ni la salida que realice una rutina con transacciones propias de DB2. Así, por ejemplo, si una UDF graba en un archivo durante una transacción y más tarde se retira dicha transacción por cualquier motivo, no se realiza ningún intento de descubrir ni deshacer las grabaciones efectuadas en el archivo.
- En las rutinas no se pueden ejecutar sentencias ni mandatos relacionados con conexiones, que incluyen:
  - BACKUP
  - CONNECT
  - CONNECT TO
  - CONNECT RESET
  - CREATE DATABASE
  - DROP DATABASE
  - FORWARD RECOVERY
  - RESTORE
- En general, DB2 no restringe el uso de funciones del sistema operativo. No obstante, existen unas excepciones:
  1. **Es imperativo que ninguna rutina instale sus propios manejadores de señales.** Si no se cumple con esta restricción, se pueden producir anomalías inesperadas, terminaciones anormales de base de datos u otros problemas. La instalación de manejadores de señales también puede interferir en el funcionamiento de la JVM para las rutinas de Java.
  2. Las llamadas al sistema que terminan un proceso pueden terminar de forma anormal uno de los procesos de DB2 y tener como consecuencia una anomalía del sistema o de la aplicación.

     Otras llamadas al sistema también pueden ocasionar problemas si interfieren en el funcionamiento normal de DB2; por ejemplo, una UDF que intente descargar una biblioteca que contenga una UDF desde la memoria podría causar problemas graves. Sea cuidadoso cuando codifique y pruebe las rutinas que contengan llamadas al sistema.
- Las rutinas no deben contener mandatos que terminen el proceso actual. Una rutina siempre tiene que devolver el control a DB2 sin terminar el proceso actual.
- Cuando se devuelvan conjuntos de resultados desde procedimientos almacenados anidados, podrá abrir un cursor con el mismo nombre en varios niveles de anidamiento. No obstante, las aplicaciones anteriores a la versión 8 sólo podrán acceder al primer conjunto de resultados que se haya abierto. Esta restricción no se aplica a los cursores abiertos con un nivel de paquete diferente.

- No cambie los cuerpos de las rutinas mientras la base de datos esté activa. Si es necesario cambiar el cuerpo de una rutina sin detener y reiniciar DB2, cree el nuevo cuerpo de la rutina con otro nombre de biblioteca. Luego puede utilizar la sentencia ALTER para cambiar el nombre externo (EXTERNAL NAME) de la rutina de forma que haga referencia al nuevo cuerpo.
- Los valores de todas las variables de entorno cuyos nombres empiezan por 'DB2' se capturan en el momento en que se inicia el gestor de bases de datos con `db2start` y están disponibles en todas las rutinas, tanto si son FENCED como NOT FENCED. La única excepción es la variable de entorno `DB2CKPTR`. Otras variables de entorno son accesibles desde rutinas NOT FENCED, pero no desde el proceso de las rutinas FENCED (por ejemplo, `LIBPATH`). Observe que las variables de entorno se *capturan*. Cualquier cambio que se realice sobre las variables de entorno después de emitir `db2start` no estará disponible para las rutinas.
- Cuando utilice recursos protegidos (recursos que sólo permiten que acceda un proceso cada vez dentro de las rutinas), debe tratar de evitar puntos muertos entre rutinas. Si se producen puntos muertos en dos o más rutinas, DB2 no será capaz de detectar o resolver la condición, lo que ocasionará que los procesos de rutinas se queden colgados.
- Si asigna memoria dinámica en una rutina, la debe liberar antes de volver a DB2. De no hacerlo, se producirá una fuga de memoria y un crecimiento continuado de los procesos de DB2, circunstancias que eventualmente podrían conducir a condiciones de falta de memoria.

  Para las UDF y los métodos, se puede utilizar el recurso del área reutilizable para anclar la memoria dinámica necesaria a lo largo de varias invocaciones. Si emplea el área reutilizable de esta manera, especifique el atributo FINAL CALL en la sentencia CREATE para la UDF o el método, a fin de que pueda liberar la memoria asignada en el proceso de fin de sentencia.
- No asigne almacenamiento para ningún parámetro de la rutina en el servidor de bases de datos. El gestor de bases de datos asigna automáticamente almacenamiento basándose en la declaración de parámetros de la sentencia CREATE. No modifique ningún puntero del almacenamiento para los parámetros de la rutina. Un intento de cambiar un puntero por un puntero del almacenamiento creado localmente puede tener como consecuencia fugas de memoria, corrupción de los datos o terminaciones anormales.
- No utilice datos estáticos ni globales en las rutinas. DB2 no puede garantizar que la memoria utilizada por las variables estáticas o globales no se toque entre las invocaciones de rutinas. Para las UDF y los métodos, puede emplear áreas reutilizables a fin de almacenar los valores que se utilizarán entre las invocaciones.
- Los valores de todos los argumentos de SQL se colocan en el almacenamiento intermedio. Esto significa que se realiza una copia del valor y se presenta a la rutina. Si se efectúan cambios en los parámetros de entrada de una rutina, estos cambios no repercutirán en los valores de SQL ni en el proceso. Sin embargo, si una rutina graba, en un parámetro de entrada o de salida, más datos de los especificados en la sentencia CREATE, se habrá corrompido la memoria y es posible que la rutina termine anormalmente.

Conceptos relacionados:

- “Consideraciones sobre el rendimiento para crear rutinas” en la página 24
- “Consideraciones sobre seguridad para las rutinas” en la página 27
- “Manejo de tipos de datos de SQL en rutinas C/C++” en la página 171

Información relacionada:

- "Sentencia CALL" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE FUNCTION" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE PROCEDURE" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE METHOD" en la publicación *Consulta de SQL, Volumen 2*
- "Tipos de datos SQL soportados en C y C++" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente*
- "Correlaciones de tipos de datos entre DB2 y OLE DB" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente*
- "Sentencia ALTER FUNCTION" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia ALTER METHOD" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia ALTER PROCEDURE" en la publicación *Consulta de SQL, Volumen 2*
- "Tipos de datos de SQL soportados en OLE DB" en la página 206
- "Tipos de datos de SQL soportados en la automatización de OLE" en la página 197

Creación de rutinas en la base de datos

Se crea una rutina en la base de datos cuando se ejecuta la sentencia CREATE para esa rutina. En el caso de las rutinas externas, la ejecución de la sentencia CREATE no sólo define el nombre y propiedades de la rutina, sino que también, con la inclusión de la cláusula EXTERNAL, apunta a la ubicación de la biblioteca del lenguaje externo que contiene la lógica de la rutina. Una rutina no se puede invocar hasta que se ha creado en la base de datos. Una rutina externa no se puede invocar satisfactoriamente hasta que se ha creado en la base de datos y hasta que la biblioteca asociada con la rutina se ha colocado en la ubicación especificada por la cláusula EXTERNAL.

Para que la rutina funcione correctamente, es fundamental crearla con las cláusulas aplicables que reflejen las características de la rutina. Son comunes muchas de las cláusulas que registran los distintos tipos de rutinas.

Requisitos previos:

A fin de conocer la lista de los privilegios necesarios para crear una rutina en la base de datos, vea las sentencias siguientes:

- CREATE FUNCTION
- CREATE METHOD
- CREATE TYPE
- CREATE PROCEDURE

Procedimiento:

Para crear una rutina, emita la sentencia CREATE con las cláusulas aplicables que correspondan al tipo de rutina con el que va a trabajar. Las sentencias son las siguientes: CREATE FUNCTION, CREATE METHOD, CREATE TYPE y CREATE PROCEDURE.

Para crear un método, tiene que haber ejecutado una sentencia CREATE TYPE, que crea un tipo estructurado. La sentencia CREATE TYPE contiene una cláusula METHOD opcional que sirve para especificar, si lo desea, una declaración de método que se asociará con el tipo. Como alternativa, puede ejecutar la sentencia

ALTER TYPE para declarar un método de un tipo estructurado existente. Después, el método se crea formalmente ejecutando la sentencia CREATE METHOD. La sentencia CREATE METHOD sólo está destinada a los atributos que tienen relación con la signatura de un método.

Una vez creada la rutina, puede invocarla desde cualquier interfaz que dé soporte a la invocación de rutinas del tipo de rutina determinado. Según el tipo de rutina, se pueden incluir: aplicaciones cliente, otras rutinas, activadores y sentencias de SQL.

Conceptos relacionados:

- "Rutinas en el desarrollo de aplicaciones" en la página 3
- "Estilos de parámetros para rutinas externas" en la página 95
- "Consideraciones sobre el rendimiento para crear rutinas" en la página 24
- "Consideraciones sobre seguridad para las rutinas" en la página 27

Tareas relacionadas:

- "Escritura de rutinas" en la página 36

Información relacionada:

- "Sentencia CREATE FUNCTION" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE PROCEDURE" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE TYPE (Estructurado)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE METHOD" en la publicación *Consulta de SQL, Volumen 2*

Ejemplos relacionados:

- "spcreate.db2 -- How to catalog the stored procedures contained in spserver.sqc (C)"
- "spserver.db2 -- To create a set of SQL procedures "
- "UDFsCreate.db2 -- How to catalog the UDFs contained in UDFsqlsv.java "

Escritura de rutinas

Los tres tipos de rutinas (procedimientos, UDF y métodos) tienen mucho en común con respecto a cómo se escriben. Por ejemplo, los tres tipos de rutinas emplean algunos estilos de parámetros iguales, soportan el uso de SQL mediante diversas interfaces de cliente (SQL incorporado, CLI y JDBC) y todos pueden invocar a otras rutinas. Con este propósito, los pasos que siguen representan un simple enfoque para escribir rutinas.

Existen algunas características de rutinas que son específicas de un tipo de rutina. Por ejemplo, los conjuntos de resultados son específicos de los procedimientos almacenados, y las áreas reutilizables son específicas de las UDF y de los métodos. Cuando llegue a un paso que no sea aplicable al tipo de rutina que ha de crear, diríjase al siguiente paso.

Requisitos previos:

Antes de escribir una rutina, debe decidir lo siguiente:

- El tipo de rutina que necesita. (Consulte el apartado Tipos de rutinas (procedimientos, funciones, métodos).)

- El lenguaje de programación que utilizará para escribirla. (Consulte el apartado Lenguajes de programación de rutinas soportados.)
- Qué interfaz debe utilizar si necesita sentencias de SQL en la rutina. (Consulte el apartado Cuándo utilizar DB2 CLI o SQL incorporado.)

Consulte también los temas relativos a consideraciones sobre seguridad, gestión de bibliotecas y clases y rendimiento.

Procedimiento:

Para crear el cuerpo de una rutina, debe hacer lo siguiente:

1. *Este punto sólo es aplicable a las rutinas externas.* Aceptar los parámetros de entrada de la aplicación o rutina que realiza la invocación y declarar los parámetros de salida. La forma en que una rutina acepta los parámetros depende del estilo de parámetros con el que cree la rutina. Cada estilo de parámetros define el conjunto de parámetros que se pasan al cuerpo de la rutina y el orden en que se pasan.

   Por ejemplo, la siguiente es una signatura del cuerpo de una UDF escrito en C (utilizando sqludf.h) para PARAMETER STYLE SQL:

   ```
   SQL_API_RC SQL_API_FN product ( SQLUDF_DOUBLE *in1,
                                   SQLUDF_DOUBLE *in2,
                                   SQLUDF_DOUBLE *outProduct,
                                   SQLUDF_NULLIND *in1NullInd,
                                   SQLUDF_NULLIND *in2NullInd,
                                   SQLUDF_NULLIND *productNullInd,
                                   SQLUDF_TRAIL_ARGS )
   ```

2. Añadir la lógica que la rutina debe ejecutar. Algunas características que puede emplear en el cuerpo de las rutinas son las siguientes:
   - Llamar a otras rutinas (anidamiento) o llamar a la rutina actual (repetición).
   - En las rutinas que se definen para que contengan SQL (CONTAINS SQL, READS SQL o MODIFIES SQL), la rutina puede emitir sentencias de SQL. Los tipos de sentencias que se pueden invocar se controlan por el modo en que se registran las rutinas.
   - En los métodos y las UDF externas, utilice áreas reutilizables para guardar el estado entre una rutina y la siguiente.
   - En los procedimientos de SQL, utilice manejadores de condiciones para determinar el comportamiento del procedimiento de SQL cuando se produzca una condición especificada. Puede definir condiciones en base a los valores de SQLSTATE.
3. *Este punto sólo es aplicable a los procedimientos almacenados.* Devolver uno o más conjuntos de resultados. Además de los parámetros individuales que se intercambian con la aplicación de llamada, los procedimientos almacenados tienen la posibilidad de devolver varios conjuntos de resultados. Sólo las rutinas de SQL y las rutinas de CLI, ODBC, JDBC y SQLJ y los clientes pueden aceptar conjuntos de resultados.

Para poder invocar una rutina, además de escribirla la tiene que registrar. Esto se hace mediante la sentencia CREATE que coincida con el tipo de rutina que se va a crear. En general, no importa el orden en que se escribe y registra la rutina. Sin embargo, el registro de una rutina debe preceder a su creación si ésta emite SQL que hace referencia a sí misma. En este caso, para que un enlace resulte satisfactorio, antes se tiene que haber producido el registro de la rutina.

Conceptos relacionados:

- "Cuándo utilizar DB2 CLI o SQL incorporado" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente*
- "Estilos de parámetros para rutinas externas" en la página 95
- "Consideraciones sobre el rendimiento para crear rutinas" en la página 24
- "Consideraciones sobre seguridad para las rutinas" en la página 27
- "Rutinas C/C++" en la página 164
- "Rutinas Java" en la página 181
- "Restricciones del uso de rutinas" en la página 32
- "Consideraciones sobre la gestión de bibliotecas y clases" en la página 30
- "Diseño de rutinas de automatización de OLE" en la página 194
- "Funciones de tabla de OLE DB definidas por el usuario" en la página 201
- "Lenguajes de programación de rutinas soportados" en la página 21

Información relacionada:

- "Sentencia CREATE FUNCTION" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE PROCEDURE" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE TYPE (Estructurado)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE METHOD" en la publicación *Consulta de SQL, Volumen 2*

Ejemplos relacionados:

- "spserver.c -- Definition of various types of stored procedures"
- "spserver.db2 -- To create a set of SQL procedures "
- "spserver.sqc -- Definition of various types of stored procedures (C)"
- "spserver.sqC -- Definition of various types of stored procedures (C++)"
- "SpServer.java -- Provide a variety of types of stored procedures to be called from (JDBC)"
- "SpServer.sqlj -- Provide a variety of types of stored procedures to be called from (SQLj)"

Autorizaciones y enlace de rutinas que contienen SQL

Al tratar la autorización al nivel de las rutinas, es importante definir algunas funciones relacionadas con éstas, la determinación de las funciones y los privilegios relacionados con las funciones:

Propietario del paquete
: El propietario de un paquete en particular que participa en la implementación de una rutina. El propietario del paquete es el usuario que ejecuta el mandato BIND para enlazar un paquete con una base de datos, a menos que se utilice la opción OWNER de precompilación/enlace (BIND) para alterar temporalmente la propiedad del paquete y se establezca en otro usuario. Después de la ejecución del mandato BIND, se otorga el privilegio EXECUTE WITH GRANT sobre el paquete al propietario del paquete. Una biblioteca de rutinas o un ejecutable puede comprender varios paquetes y, por lo tanto, puede tener asociados varios propietarios de paquetes.

Definidor de la rutina
: El ID que emite la sentencia CREATE para registrar una rutina. Generalmente, el definidor de la rutina es un DBA, pero, a menudo,

también es el propietario del paquete de la rutina. Cuando se invoca una rutina, durante la carga de los paquetes, la autorización para ejecutarla se comprueba con la autorización del definidor para ejecutar el paquete o paquetes asociados con la rutina (no con la autorización del invocador de la rutina). A fin de que una rutina se invoque satisfactoriamente, el definidor de la misma debe disponer del siguiente privilegio o autorización:

- Privilegio EXECUTE sobre el paquete o paquetes de la rutina y privilegio EXECUTE sobre la rutina
- Autorización SYSADM o DBADM

Si el definidor de la rutina y el propietario del paquete de la rutina son el mismo usuario, el definidor de la rutina tendrá los privilegios EXECUTE necesarios sobre los paquetes. Si el definidor no es el propietario del paquete, debe otorgar explícitamente el privilegio EXECUTE sobre los paquetes al definidor el propietario del paquete o cualquier usuario con la autorización SYSADM o DBADM.

Después de emitir la sentencia CREATE que registra la rutina, se otorga implícitamente al definidor el privilegio EXECUTE WITH GRANT OPTION sobre la rutina.

La función del definidor de la rutina es encapsular, bajo un ID de autorización, los privilegios de ejecutar los paquetes asociados con una rutina y el privilegio de otorgar el privilegio EXECUTE sobre la rutina a PUBLIC o a usuarios específicos que tengan que invocar la rutina.

Nota: Para las rutinas de SQL, el definidor de la rutina también es implícitamente el propietario del paquete. Por consiguiente, el definidor tendrá el privilegio EXECUTE WITH GRANT OPTION sobre la rutina y sobre el paquete de la rutina después de la ejecución de la sentencia CREATE para la rutina.

Invocador de la rutina
El ID que invoca la rutina. Para determinar qué usuarios serán invocadores de una rutina, es necesario tomar en consideración cómo se puede invocar una rutina. Las rutinas se pueden invocar desde una ventana de mandatos o desde una aplicación de SQL intercalado. En el caso de los métodos y UDF, la referencia a la rutina estará intercalada en otra sentencia de SQL. Un procedimiento se invoca utilizando la sentencia CALL. Para el SQL dinámico de una aplicación, el invocador es el ID de autorización de ejecución de la rutina de nivel inmediatamente superior o aplicación que contiene la invocación de la rutina (no obstante, este ID también puede depender de la opción DYNAMICRULES con la que se ha enlazado la rutina de nivel superior o aplicación). Para el SQL estático, el invocador es el valor de la opción OWNER de precompilación/enlace (BIND) del paquete que contiene la referencia a la rutina. Con el fin de invocar la rutina satisfactoriamente, estos usuarios necesitarán el privilegio EXECUTE sobre la rutina. Este privilegio puede ser otorgado por cualquier usuario con el privilegio EXECUTE WITH GRANT OPTION sobre la rutina (esto incluye el definidor de la rutina, a menos que el privilegio se haya revocado explícitamente) o la autorización SYSADM o DBADM emitiendo explícitamente una sentencia GRANT.

Como ejemplo, si un paquete asociado con una aplicación que contiene SQL dinámico se ha enlazado con DYNAMICRULES BIND, su ID de autorización de

ejecución será el propietario del paquete, no la persona que invoque el paquete. Asimismo, el propietario del paquete será el enlazador real o el valor de la opción de precompilación/enlace OWNER. En este caso, el invocador de la rutina asume este valor en lugar del ID del usuario que está ejecutando la aplicación.

Notas:

1. Para el SQL estático contenido en una rutina, los privilegios del propietario del paquete deben bastar para ejecutar las sentencias de SQL del cuerpo de la rutina. Estas sentencias de SQL pueden requerir privilegios de acceso a tablas o de ejecución si hay referencias anidadas a rutinas.
2. Para el SQL dinámico contenido en una rutina, los privilegios del ID de usuario que se validarán se gobiernan mediante la opción DYNAMICRULES de BIND del cuerpo de la rutina.
3. El propietario del paquete de la rutina debe tener el privilegio GRANT EXECUTE sobre el paquete para el definidor de la rutina. Esto se puede efectuar antes o después de que se registre la rutina, pero es necesario antes de que ésta se invoque, pues, de lo contrario, se devolverá un error (SQLSTATE 42051).

Los pasos implicados en la gestión del privilegio de ejecución sobre una rutina se detallan en el diagrama y texto que siguen:

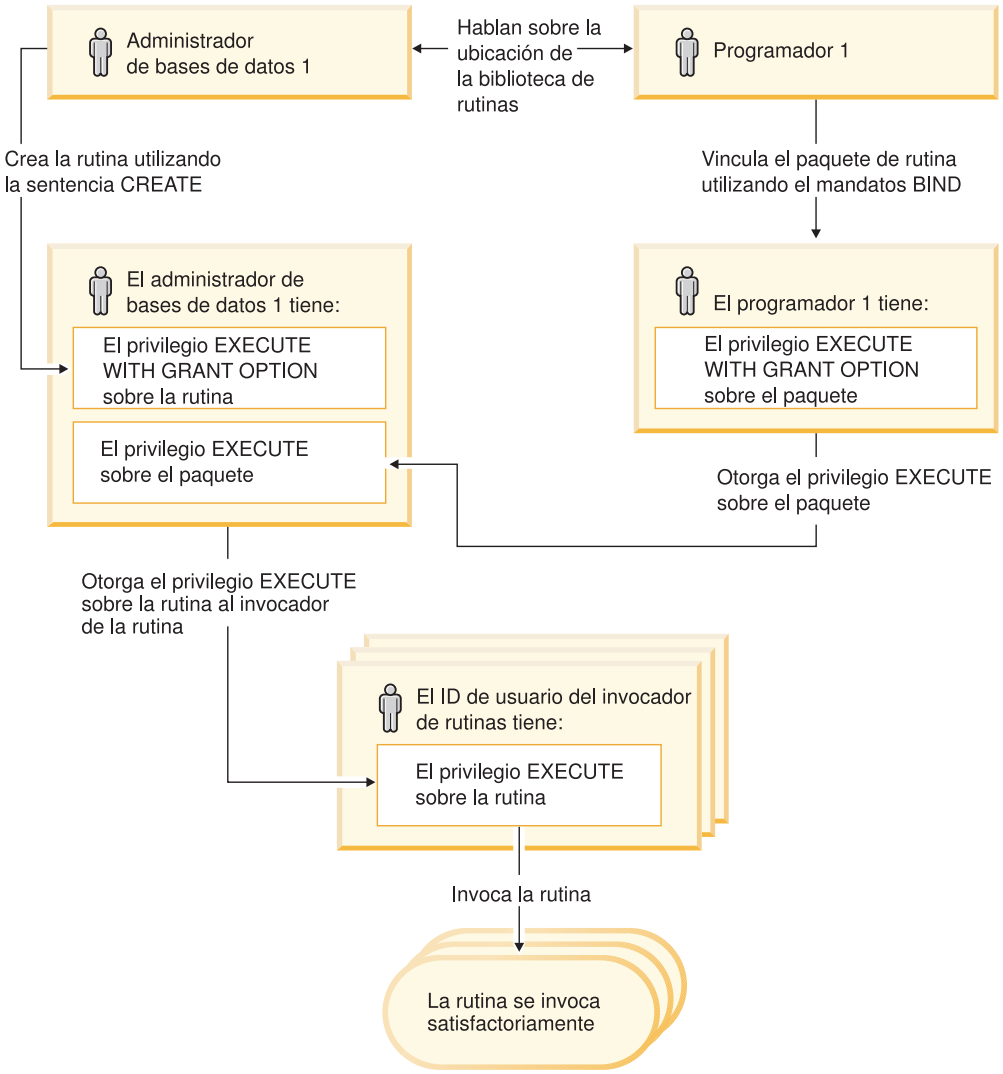


Figura 2. Gestión del privilegio EXECUTE sobre las rutinas

1. El definidor ejecuta la sentencia CREATE adecuada para registrar la rutina. Esta acción registra la rutina en DB2® con su nivel deseado de acceso de SQL, establece la signatura de la rutina y también apunta al ejecutable de la rutina. El definidor, si no es también el propietario del paquete, se tiene que comunicar con los propietarios de paquetes y autores de los programas de rutinas para que quede claro dónde residen las bibliotecas de rutinas, a fin de que se pueda especificar correctamente en la cláusula EXTERNAL de la sentencia CREATE. En virtud de una sentencia CREATE satisfactoria, el definidor tiene el privilegio EXECUTE WITH GRANT sobre la rutina, pero todavía no tiene el privilegio EXECUTE sobre los paquetes de la rutina.
2. El definidor debe otorgar el privilegio EXECUTE sobre la rutina a los usuarios que tengan permitido utilizar la rutina. (Si el paquete para esta rutina la va a llamar de forma repetitiva, se tiene que realizar este paso antes que el siguiente.)
3. Los propietarios de paquetes precompilan y enlazan el programa de rutina o hacen que otros realicen estas acciones de su parte. Después de una precompilación y enlace satisfactorios, se otorga implícitamente al propietario del paquete el privilegio EXECUTE WITH GRANT OPTION sobre el paquete respectivo. Este paso sigue al primer paso de esta lista, únicamente para

abarcar la posibilidad de una repetición de SQL en la rutina. Si tal repetición no existe en ningún caso en particular, la precompilación/enlace puede preceder a la emisión de la sentencia CREATE para la rutina.

4. Cada propietario de un paquete debe otorgar explícitamente al definidor de la rutina el privilegio EXECUTE sobre el paquete de rutina respectivo. Este paso se tiene que llevar a cabo en algún momento posterior al paso anterior. Si el propietario del paquete también es el definidor de la rutina, este paso se puede omitir.
5. Uso estático de la rutina: al propietario de enlace del paquete que hace referencia a la rutina se le tiene que haber asignado el privilegio EXECUTE sobre la rutina, por lo que en este punto se tiene que haber efectuado el paso anterior. Cuando se ejecuta la rutina, DB2 verifica que el definidor tiene el privilegio EXECUTE sobre los paquetes que sean necesarios, por lo que se debe llevar a cabo el paso 3 para cada uno de dichos paquetes.
6. Uso dinámico de la rutina: el ID de autorización, controlado mediante la opción DYNAMICRULES, para la aplicación que realiza la invocación debe tener el privilegio EXECUTE sobre la rutina (paso 4), y el definidor de la rutina debe tener el privilegio EXECUTE sobre los paquetes (paso 3).

Conceptos relacionados:

- "Privileges, authority levels, and database authorities" en la publicación *Administration Guide: Implementation*
- "Efecto de la opción de vinculación DYNAMICRULES en SQL dinámico" en la página 113
- "Routine privileges" en la publicación *Administration Guide: Implementation*

Información relacionada:

- "Sentencia CREATE FUNCTION" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE PROCEDURE" en la publicación *Consulta de SQL, Volumen 2*
- "Mandato BIND" en la publicación *Consulta de mandatos*

Depuración de rutinas

Antes de desplegar las rutinas en un servidor de producción, debe probarlas y depurarlas a fondo en un servidor de prueba. Esto es especialmente importante para las rutinas que se han de registrar como NOT FENCED porque tienen acceso no restringido a la memoria del gestor de bases de datos, a las bases de datos de éste y a las estructuras de control de las bases de datos. Las rutinas FENCED THREADSAFE también exigen una gran atención, ya que comparten memoria con otras rutinas.

Procedimiento:

Lista de comprobación de problemas habituales de las rutinas

Para asegurarse de que una rutina se ejecuta correctamente, realice las comprobaciones siguientes:

- La rutina se ha registrado correctamente. Los parámetros proporcionados en la sentencia CREATE deben coincidir con los argumentos manejados por el cuerpo de la rutina. Teniendo esto en cuenta, compruebe los siguientes elementos específicos:
  - Los tipos de datos de los argumentos utilizados por el cuerpo de la rutina son adecuados para los tipos de parámetros definidos en la sentencia CREATE.

- La rutina no graba, en una variable de salida, más bytes de los definidos para el resultado correspondiente de la sentencia CREATE.
- Los argumentos de rutina para SCRATCHPAD, FINAL CALL y DBINFO están presentes si la rutina se ha registrado con las opciones de CREATE que corresponden.
- En las rutinas externas, el valor de la cláusula EXTERNAL NAME de la sentencia CREATE debe coincidir con la biblioteca y punto de entrada de las rutinas (según el sistema operativo, será sensible a las mayúsculas y minúsculas o no).
- En las rutinas C++, el compilador C++ aplica la decoración de tipos al nombre de punto de entrada. Hay que especificar el nombre decorado con tipos en la cláusula EXTERNAL NAME o bien se debe definir el punto de entrada como `extern "C"` en el código del usuario.
- El nombre de rutina especificado durante la invocación debe coincidir con el nombre registrado (definido en la sentencia CREATE) de la rutina. Por omisión, los identificadores de rutina se convierten a mayúsculas. Esto no se aplica a los identificadores delimitados, que no se convierten a mayúsculas y que, por lo tanto, son sensibles a las mayúsculas y minúsculas.

  La rutina debe estar ubicada en la vía de acceso de directorio especificada en la sentencia CREATE o, si no se facilita una vía de acceso, donde DB2 la busque por omisión. Para las UDF, métodos y procedimientos protegidos, es la siguiente: `sqllib/function` (UNIX) o `sqllib\function` (Windows). Para los procedimientos no protegidos, es la siguiente: `sqllib/function/unfenced` (UNIX) o `sqllib\function\unfenced` (Windows).

- La rutina se ha construido utilizando las opciones correctas de secuencia de llamada, precompilación (si tiene SQL incorporado), compilación y enlace.
- La aplicación se encuentra vinculada a la base de datos, excepto si se ha escrito utilizando la CLI de DB2, ODBC o JDBC. La rutina también se debe vincular si contiene SQL y no utiliza ninguna de estas interfaces.
- La rutina devuelve a la aplicación cliente cualquier información sobre errores de manera exacta.
- Todos los tipos de llamadas aplicables se tienen en cuenta si la rutina se ha definido con FINAL CALL.
- Se devuelven los recursos del sistema utilizados por las rutinas.
- Si intenta invocar una rutina y recibe un error (SQLCODE -551, SQLSTATE 42501) que indica que no tiene suficientes privilegios para realizar esta operación, es probable que le falte el privilegio EXECUTE sobre la rutina. El que puede otorgar este privilegio a cualquier invocador de una rutina es un usuario con autorización SYSADM, DBADM o bien el definidor de la rutina. El tema relacionado con autorizaciones y rutinas proporciona los detalles sobre cómo gestionar de forma eficaz el uso de este privilegio.

Técnicas de depuración de las rutinas

Para depurar una rutina, utilice las técnicas siguientes:

- El Centro de desarrollo brinda extensas herramientas de depuración para los procedimientos Java e incorporados al SQL.
- No es posible escribir datos de diagnóstico en pantalla desde una rutina. Si va a grabar datos de diagnóstico en un archivo, asegúrese de que la

grabación se efectúe en un directorio globalmente accesible, como \tmp. No grabe en los directorios utilizados por bases de datos o gestores de bases de datos.

Para los procedimientos, una alternativa segura es grabar datos de diagnóstico en una tabla de SQL. El procedimiento que se está probando debe estar registrado con la cláusula MODIFIES SQL DATA para poder grabar en una tabla de SQL. Si se necesita un procedimiento existente para grabar datos (o dejar de grabar datos) en una tabla de SQL, se debe eliminar y volver a registrar el procedimiento con (o sin) la cláusula MODIFIES SQL DATA. Antes de eliminar y volver a registrar el procedimiento, debe saber cuáles son sus dependencias.

- Puede depurar la rutina localmente escribiendo una simple aplicación que invoque de forma directa al punto de entrada de la rutina. Para obtener información sobre cómo utilizar el depurador suministrado, consulte la documentación del compilador.

Conceptos relacionados:

- "Autorizaciones y enlace de rutinas que contienen SQL" en la página 38
- "Consideraciones sobre seguridad para las rutinas" en la página 27

Tareas relacionadas:

- "Depuración de procedimientos almacenados de Java" en la página 189
- "Devolución de mensajes de error desde procedimientos de SQL" en la página 77

Información relacionada:

- "Identificadores" en la publicación *Consulta de SQL, Volumen 1*
- "Sentencia CREATE FUNCTION" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE PROCEDURE" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE TYPE (Estructurado)" en la publicación *Consulta de SQL, Volumen 2*
- "Mandato BIND" en la publicación *Consulta de mandatos*
- "Mandato PRECOMPILE" en la publicación *Consulta de mandatos*
- "Sentencia CREATE METHOD" en la publicación *Consulta de SQL, Volumen 2*
- "Tipos de datos de SQL soportados en OLE DB" en la página 206
- "Sintaxis para pasar argumentos a rutinas escritas en C/C++, OLE o COBOL" en la página 97
- "Tipos de datos de SQL soportados en la automatización de OLE" en la página 197
- "Tipos de datos de SQL soportados en C/C++" en la página 168

Conflictos de datos cuando los procedimientos leen o graban en tablas

Para conservar la integridad de la base de datos, es necesario evitar conflictos al leer y grabar en tablas. Por ejemplo, suponga que una aplicación está actualizando la tabla EMPLOYEE y que la sentencia llama a una rutina. Suponga que la rutina intenta leer la tabla EMPLOYEE y encuentra la fila que la aplicación está actualizando. La fila tiene un estado indeterminado desde la perspectiva de la rutina: quizá algunas columnas de la fila se han actualizado y otras no. Si la rutina

actúa sobre esta fila parcialmente actualizada, puede emprender acciones incorrectas. Para evitar esta clase de problemas, DB2® no permite operaciones que entren en conflicto en ninguna tabla.

Para describir cómo DB2 evita los conflictos al leer y grabar en tablas desde las rutinas, son necesarios los dos términos siguientes:

sentencia de nivel superior
Una sentencia de nivel superior es cualquier sentencia de SQL emitida desde una aplicación o desde un procedimiento almacenado que se haya invocado como sentencia de nivel superior. Si un procedimiento se invoca dentro de una sentencia dinámica compuesta o un activador, la sentencia compuesta o la sentencia que desencadena el activador es la sentencia de nivel superior. Si una función o método de SQL contiene una sentencia CALL anidada, la sentencia que invoca a la función o al método es la sentencia de nivel superior.

contexto de acceso a una tabla
El contexto de acceso a una tabla hace referencia al ámbito en que se permiten operaciones conflictivas sobre una tabla. Se crea un contexto de acceso a una tabla siempre que:

- Una sentencia de nivel superior emite una sentencia de SQL.
- Se invoca una UDF o un método.
- Se invoca un procedimiento desde un activador, una sentencia dinámica compuesta, una función de SQL o un método de SQL.

Por ejemplo, cuando una aplicación llama a un procedimiento almacenado, CALL es una sentencia de nivel superior y, por lo tanto, obtiene un contexto de acceso a una tabla. Si el procedimiento almacenado efectúa una actualización (UPDATE), UPDATE es también una sentencia de nivel superior (puesto que el procedimiento almacenado se ha invocado como sentencia de nivel superior), por lo cual obtiene un contexto de acceso a una tabla. Si UPDATE invoca una UDF, la UDF obtendrá un contexto de acceso a una tabla por separado y las sentencias de SQL dentro de la UDF no serán sentencias de nivel superior.

Una vez que se ha accedido a una tabla para la lectura o grabación, está protegida de conflictos en la sentencia de nivel superior que ha efectuado el acceso. La tabla se puede leer o grabar desde una sentencia de nivel superior distinta o desde una rutina invocada desde una sentencia de nivel superior distinta.

Se aplican las normas siguientes:

1. En un contexto de acceso a una tabla, una tabla determinada se puede leer y grabar sin causar un conflicto.
2. Si se está leyendo una tabla en un contexto de acceso a una tabla, otros contextos también pueden leerla. Sin embargo, si cualquier otro contexto intenta grabar en la tabla, se producirá un conflicto.
3. Si se está grabando en una tabla en un contexto de acceso a una tabla, ningún otro contexto puede leer o grabar en ella sin causar un conflicto.

Si se produce un conflicto, se devuelve un error (SQLCODE -746, SQLSTATE 57053) a la sentencia que lo ha ocasionado.

A continuación se muestra un ejemplo de conflictos de lectura y grabación de tablas:

Suponga que una aplicación emite la sentencia siguiente:

```
UPDATE t1 SET c1 = udf1(c2)
```

La UDF1 contiene las sentencias siguientes:

```
DECLARE cur1 CURSOR FOR SELECT c1, c2 FROM t1
OPEN cur1
```

Esto tendrá como consecuencia un conflicto, puesto que se viola la norma 3. Este tipo de conflicto sólo se puede resolver rediseñando la aplicación o la UDF.

La situación siguiente no producirá un conflicto:

Suponga que una aplicación emite las sentencias siguientes:

```
DECLARE cur2 CURSOR FOR SELECT udf2(c1) FROM t2
OPEN cur2
FETCH cur2 INTO :hv
UPDATE t2 SET c2 = 5
```

La UDF2 contiene las sentencias siguientes:

```
DECLARE cur3 CURSOR FOR SELECT c1, c2 FROM t2
OPEN cur3
FETCH cur3 INTO :hv
```

Con el cursor, la UDF2 tiene permitido leer la tabla T2, puesto que dos contextos de acceso a una tabla pueden leer la misma tabla. La aplicación tiene permitido actualizar la tabla T2 aunque la UDF2 esté leyendo la tabla, porque la UDF2 se ha invocado en una sentencia de nivel de aplicación distinta de la actualización.

Conceptos relacionados:

- "Rutinas en el desarrollo de aplicaciones" en la página 3
- "SQL en rutinas externas" en la página 110

Características de los procedimientos

Los procedimientos almacenados tienen posibilidades especiales para el intercambio de datos con las aplicaciones y rutinas que realizan la invocación. Los apartados siguientes describen las modalidades de parámetros de procedimiento, la posibilidad de los procedimientos de devolver conjuntos de resultados y la opción de aceptar parámetros con el estilo de una rutina principal o una subrutina.

Modalidades de parámetros de procedimiento

Las aplicaciones cliente y las rutinas de llamada intercambian información con los procedimientos a través de parámetros y conjuntos de resultados. Los parámetros para las rutinas se definen con tipos de datos específicos. A diferencia de otras rutinas, los parámetros de los procedimientos también se definen por la dirección en que viajan los datos (la modalidad de parámetro).

Existen tres tipos de parámetros para los procedimientos:

- Parámetros IN: los datos pasados al procedimiento.
- Parámetros OUT: los datos devueltos por el procedimiento.
- Parámetros INOUT: los datos pasados al procedimiento que, durante la ejecución del mismo, se sustituyen por datos que el procedimiento debe devolver.

La modalidad de los parámetros y sus tipos de datos se definen cuando se registra un procedimiento con la sentencia CREATE PROCEDURE.

Conceptos relacionados:

- "Conjuntos de resultados de procedimiento" en la página 47

Tareas relacionadas:

- "Creación de rutinas en la base de datos" en la página 35

Información relacionada:

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

Conjuntos de resultados de procedimiento

Los apartados siguientes describen cómo se pueden devolver conjuntos de resultados con los procedimientos, y muestran la forma de devolver y recibir conjuntos de resultados utilizando varias interfaces.

Conjuntos de resultados de procedimiento

Además de intercambiar parámetros, los procedimientos pueden pasar información a los invocadores devolviendo conjuntos de resultados. Los conjuntos de resultados se pueden aceptar en las rutinas incorporadas al SQL y en las rutinas y aplicaciones programadas en las interfaces siguientes:

- CLI
- JDBC
- SQLJ
- ODBC

Los procedimientos almacenados pasan conjuntos de resultados a sus invocadores mediante los cursores. El cuerpo del procedimiento debe contener un cursor para cada conjunto de resultados que sea necesario devolver. Mientras que se pueden captar filas desde un cursor de conjunto de resultados dentro del procedimiento, sólo las filas no captadas pasan al invocador como el conjunto de resultados. Al salir de un procedimiento, deje abiertos los cursores que correspondan a los conjuntos de resultados. Los diversos conjuntos de resultados se devuelven en el orden en que se abren sus cursores.

Al declarar un cursor para un conjunto de resultados, es muy recomendable que especifique el destino en la cláusula WITH RETURN TO de la sentencia DECLARE CURSOR (en los procedimientos de SQL, esto es obligatorio). Para devolver el conjunto de resultados al invocador, tanto si éste es una aplicación como si es una rutina, especifique WITH RETURN TO CALLER. Para devolver el conjunto de resultados directamente a la aplicación, ignorando cualquier rutina anidada intermedia, especifique WITH RETURN TO CLIENT. En las rutinas externas, los cursores se definen como WITH RETURN TO CALLER por omisión, a menos que se definan explícitamente como WITH RETURN TO CLIENT.

Cuando registre un procedimiento con la sentencia CREATE PROCEDURE, indique el número de conjuntos de resultados que éste devuelve mediante la cláusula DYNAMIC RESULT SETS. Este valor se encuentra en la columna RESULT_SETS de la vista SYSCAT.ROUTINES. Si el número de conjuntos de resultados devueltos desde un procedimiento es diferente del número especificado en la sentencia CREATE PROCEDURE, se emite un aviso (SQLCODE +464, SQLSTATE 0100E). Para los procedimientos almacenados PARAMETER STYLE JAVA, el número de

conjuntos de resultados de la sentencia CREATE PROCEDURE debe coincidir con el número de parámetros ResultSet[] en la signatura del método Java™.

El invocador puede describir (DESCRIBE) los conjuntos de resultados recibidos. Tenga en cuenta que, si se abre el mismo cursor en varios niveles de anidamiento, las aplicaciones que se ejecuten en los clientes DB2® UDB Versión 7 sólo podrán describir (DESCRIBE) el primer conjunto de resultados que se abra.

El invocador debe procesar en serie los conjuntos de resultados (si el invocador no es una rutina incorporada al SQL). Un cursor se abre automáticamente en el primer conjunto de resultados y se proporciona una llamada especial (`SQLMoreResults` para la CLI de DB2, `getMoreResults` para JDBC, `getNextResultSet` para SQLJ) a fin de cerrar el cursor en un conjunto de resultados y abrirlo en el siguiente.

Para recibir los conjuntos de resultados en las rutinas incorporadas al SQL, debe declarar (DECLARE) y asociar (ASSOCIATE) localizadores de conjunto de resultados con el procedimiento que espere que devolverá conjuntos de resultados. Después, debe asignar (ALLOCATE) cada cursor que espere que se devolverá a un localizador de conjunto de resultados. Una vez hecho esto, podrá captar filas desde los conjuntos de resultados.

Si se invoca un procedimiento dentro de un activador, una sentencia dinámica compuesta, una función de SQL o un método de SQL, los conjuntos de resultados no serán accesibles.

Nota: Una sentencia COMMIT emitida desde el procedimiento o la aplicación cerrará los conjuntos de resultados que no correspondan a los cursores WITH HOLD. Una sentencia ROLLBACK emitida desde la aplicación o el procedimiento almacenado cerrará todos los cursores de conjuntos de resultados. Después de que se ejecute una sentencia COMMIT o ROLLBACK desde un procedimiento, los cursores se pueden abrir y devolver como conjuntos de resultados.

Conceptos relacionados:

- "Procedimientos" en la página 13
- "Cursors in CLI applications" en la publicación *CLI Guide and Reference, Volume 1*
- "Result set terminology in CLI applications" en la publicación *CLI Guide and Reference, Volume 1*
- "Result set retrieval into arrays in CLI applications" en la publicación *CLI Guide and Reference, Volume 1*

Tareas relacionadas:

- "Declaración y utilización de cursores en programas de SQL estático" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente*
- "Declaración y utilización de cursores en programas de SQL dinámico" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente*
- "Devolución de conjuntos de resultados desde procedimientos de SQL y SQL incorporado" en la página 49
- "Recepción de conjuntos de resultados de procedimiento en las rutinas de SQL" en la página 53
- "Recepción de conjuntos de resultados de procedimiento en aplicaciones y rutinas JDBC" en la página 55

- "Devolución de conjuntos de resultados desde procedimientos JDBC" en la página 52
- "Recepción de conjuntos de resultados de procedimiento en aplicaciones y rutinas SQLJ" en la página 54
- "Devolución de conjuntos de resultados desde procedimientos de SQLJ" en la página 51
- "Devolución de conjuntos de resultados desde procedimientos CLR" en la página 125

Información relacionada:

- "Sentencia COMMIT" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE PROCEDURE" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia DESCRIBE" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia PREPARE" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia ROLLBACK" en la publicación *Consulta de SQL, Volumen 2*
- "Vista de catálogo SYSCAT.ROUTINES" en la publicación *Consulta de SQL, Volumen 1*

Devolución de conjuntos de resultados desde procedimientos de SQL y SQL incorporado

Es posible desarrollar procedimientos que devuelvan conjuntos de resultados a la rutina o aplicación que realiza la invocación. En los procedimientos de SQL y SQL incorporado, la devolución de conjuntos de resultados se maneja con la sentencia DECLARE CURSOR.

Procedimiento:

Para devolver un conjunto de resultados de un procedimiento de SQL o SQL incorporado:

1. Declare un cursor utilizando la sentencia DECLARE CURSOR. La declaración de cursor incluye la sentencia SELECT que genera el conjunto de filas que formarán el conjunto de resultados. En la declaración de cursor, se le recomienda encarecidamente que especifique el destino del conjunto de resultados con la cláusula WITH RETURN TO (es obligatorio para los procedimientos de SQL).
   - Para devolver un conjunto de resultados al invocador de un procedimiento, tanto si el invocador es una aplicación cliente como si es otra rutina, utilice la cláusula WITH RETURN TO CALLER.

     En el ejemplo siguiente, el procedimiento de SQL "CALLER_SET" utiliza la cláusula WITH RETURN TO CALLER para devolver un conjunto de resultados al invocador de CALLER_SET:

     ```
     CREATE PROCEDURE CALLER_SET()
     DYNAMIC RESULT SETS 1
     LANGUAGE SQL
     BEGIN
        DECLARE clientcur CURSOR WITH RETURN TO CALLER
            FOR SELECT name, dept, job
            FROM staff
            WHERE salary > 15000;
        OPEN clientcur;
     END
     ```
   - Para devolver un conjunto de resultados desde un procedimiento a la aplicación de origen, utilice la cláusula WITH RETURN TO CLIENT. Cuando

se especifica WITH RETURN TO CLIENT en un conjunto de resultados, no puede acceder al conjunto de resultados ningún procedimiento anidado.

En el ejemplo siguiente, el procedimiento de SQL "CLIENT_SET" utiliza la cláusula WITH RETURN TO CLIENT en la sentencia DECLARE CURSOR para devolver un conjunto de resultados a la aplicación cliente, aunque "CLIENT_SET" se invoque como rutina anidada:

```
CREATE PROCEDURE CLIENT_SET()
DYNAMIC RESULT SETS 1
LANGUAGE SQL
BEGIN
   DECLARE clientcur CURSOR WITH RETURN TO CLIENT
       FOR SELECT name, dept, job
       FROM staff
       WHERE salary > 20000;
   OPEN clientcur;
END
```

2. Abra el cursor utilizando la sentencia OPEN. Una vez abierto el cursor en el procedimiento, se podrán captar (FETCH) filas desde el mismo. Sin embargo, el conjunto de resultados que se devuelva a la aplicación o rutina de llamada únicamente contendrá filas no captadas.
3. Salga del procedimiento sin cerrar el cursor.

Si todavía no lo ha hecho, desarrolle una aplicación cliente o rutina llamante que acepte los conjuntos de resultados del procedimiento.

Conceptos relacionados:

- "Manejadores de condiciones en los procedimientos de SQL" en la página 78
- "Variables SQLCODE y SQLSTATE en procedimientos de SQL" en la página 81
- "Conjuntos de resultados de procedimiento" en la página 47

Tareas relacionadas:

- "Creación de procedimientos SQL" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Llamada a procedimientos desde el Procesador de línea de mandatos (CLP)" en la página 222
- "Llamada a procedimientos SQL desde aplicaciones cliente" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Invocación de procedimientos SQL mediante aplicaciones cliente en Windows" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Recepción de conjuntos de resultados de procedimiento en las rutinas de SQL" en la página 53
- "Recepción de conjuntos de resultados de procedimiento en aplicaciones y rutinas JDBC" en la página 55
- "Recepción de conjuntos de resultados de procedimiento en aplicaciones y rutinas SQLJ" en la página 54

Información relacionada:

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

Ejemplos relacionados:

- "spserver.sqc -- Definition of various types of stored procedures (C)"

- "spserver.sqC -- Definition of various types of stored procedures (C++)"

Devolución de conjuntos de resultados desde procedimientos de SQLJ

Es posible desarrollar procedimientos de SQLJ que devuelvan conjuntos de resultados a la rutina o aplicación que realiza la invocación. En los procedimientos de SQLJ, la devolución de conjuntos de resultados se maneja con objetos ResultSet.

Procedimiento:

Para devolver un conjunto de resultados de un procedimiento de SQLJ:

1. Declare una clase de iterador para manejar datos de consulta. Por ejemplo:
   ```
   #sql iterator SpServerEmployees(String, String, double);
   ```
2. Para cada conjunto de resultados que se ha de devolver, incluya un parámetro del tipo ResultSet[] en la declaración del procedimiento. Por ejemplo, la siguiente signatura de función acepta una matriz de objetos ResultSet:
   ```
   public static void getHighSalaries(
     double inSalaryThreshold,         // doble entrada
     int[] errorCode,                  // salida de SQLCODE
     ResultSet[] rs)                   // salida de ResultSet
   ```
3. Cree una instancia de un objeto de iterador. Por ejemplo:
   ```
   SpServerEmployees c1;
   ```
4. Asigne la sentencia de SQL que generará el conjunto de resultados a un iterador. En el ejemplo siguiente, se utiliza una variable del lenguaje principal (denominada inSalaryThreshold -- vea el ejemplo de signatura de función anterior) en la cláusula WHERE de la consulta:
   ```
   #sql c1 = {SELECT name, job, CAST(salary AS DOUBLE)
                        FROM staff
                WHERE salary > :inSalaryThreshold
                ORDER BY salary};
   ```
5. Ejecute la sentencia y obtenga el conjunto de resultados:
   ```
   rs[0] = c1.getResultSet();
   ```

Si todavía no lo ha hecho, desarrolle una aplicación cliente o rutina llamante que acepte los conjuntos de resultados del procedimiento.

Conceptos relacionados:

- "Conjuntos de resultados de procedimiento" en la página 47

Tareas relacionadas:

- "Recepción de conjuntos de resultados de procedimiento en las rutinas de SQL" en la página 53
- "Recepción de conjuntos de resultados de procedimiento en aplicaciones y rutinas JDBC" en la página 55
- "Recepción de conjuntos de resultados de procedimiento en aplicaciones y rutinas SQLJ" en la página 54

Ejemplos relacionados:

- "SpServer.sqlj -- Provide a variety of types of stored procedures to be called from (SQLj)"

Devolución de conjuntos de resultados desde procedimientos JDBC

Es posible desarrollar procedimientos JDBC que devuelvan conjuntos de resultados a la rutina o aplicación que realiza la invocación. En los procedimientos JDBC, la devolución de conjuntos de resultados se maneja con objetos ResultSet.

Procedimiento:

Para devolver un conjunto de resultados de un procedimiento JDBC:

1. Para cada conjunto de resultados que se ha de devolver, incluya un parámetro del tipo ResultSet[] en la declaración del procedimiento. Por ejemplo, la siguiente signatura de función acepta una matriz de objetos ResultSet:

```
public static void getHighSalaries(
  double inSalaryThreshold,         // doble entrada
  int[] errorCode,                  // salida de SQLCODE
  ResultSet[] rs)                   // salida de ResultSet
```

2. Abra la conexión de base de datos del invocador (utilizando un objeto Connection):

```
Connection con =
    DriverManager.getConnection("jdbc:default:connection");
```

3. Prepare la sentencia de SQL que generará el conjunto de resultados (utilizando un objeto PreparedStatement). En el ejemplo siguiente, la preparación va seguida de la asignación de una variable de entrada (denominada inSalaryThreshold - vea el ejemplo de signatura de función anterior) al valor del marcador de parámetro (un marcador de parámetro se indica con un ″?″) en la sentencia de consulta.

```
String query =
    "SELECT name, job, CAST(salary AS DOUBLE) FROM staff " +
    "  WHERE salary > ? " +
    "  ORDER BY salary";

    PreparedStatement stmt = con.prepareStatement(query);
stmt.setDouble(1, inSalaryThreshold);
```

4. Ejecute la sentencia:

```
rs[0] = stmt.executeQuery();
```

5. Finalice el cuerpo del procedimiento.

Si todavía no lo ha hecho, desarrolle una aplicación cliente o rutina llamante que acepte los conjuntos de resultados del procedimiento almacenado.

Conceptos relacionados:

- “Conjuntos de resultados de procedimiento” en la página 47

Tareas relacionadas:

- “Recepción de conjuntos de resultados de procedimiento en las rutinas de SQL” en la página 53
- “Recepción de conjuntos de resultados de procedimiento en aplicaciones y rutinas JDBC” en la página 55
- “Recepción de conjuntos de resultados de procedimiento en aplicaciones y rutinas SQLJ” en la página 54

Ejemplos relacionados:

- "SpServer.java -- Provide a variety of types of stored procedures to be called from (JDBC)"

Recepción de conjuntos de resultados de procedimiento en las rutinas de SQL

Puede recibir conjuntos de resultados de procedimientos invocados desde una rutina incorporada al SQL.

Requisitos previos:

Debe saber cuántos conjuntos de resultados devolverá el procedimiento invocado. Hay que declarar un conjunto de resultados por cada conjunto de resultados que reciba la rutina que realiza la invocación.

Procedimiento:

Para aceptar conjuntos de resultados de procedimiento desde una rutina incorporada al SQL:

1. Debe declarar (DECLARE) localizadores de conjunto de resultados para cada conjunto de resultados que devuelva el procedimiento. Por ejemplo:
   ```
   DECLARE result1 RESULT_SET_LOCATOR VARYING;
   DECLARE result2 RESULT_SET_LOCATOR VARYING;
   DECLARE result3 RESULT_SET_LOCATOR VARYING;
   ```
2. Invoque el procedimiento. Por ejemplo:
   ```
   CALL targetProcedure();
   ```
3. Asocie (ASSOCIATE) las variables de localizador de conjunto de resultados (definidas anteriormente) con el procedimiento invocado. Por ejemplo:
   ```
   ASSOCIATE RESULT SET LOCATORS(result1, result2, result3)
      WITH PROCEDURE targetProcedure;
   ```
4. Asigne (ALLOCATE) los cursores de conjunto de resultados que se han pasado desde el procedimiento invocado a los localizadores de conjunto de resultados. Por ejemplo:
   ```
   ALLOCATE rsCur CURSOR FOR RESULT SET result1;
   ```
5. Capte (FETCH) las filas desde los conjuntos de resultados. Por ejemplo:
   ```
   FETCH rsCur INTO ...
   ```

Conceptos relacionados:

- "Conjuntos de resultados de procedimiento" en la página 47

Tareas relacionadas:

- "Devolución de conjuntos de resultados desde procedimientos de SQL y SQL incorporado" en la página 49
- "Devolución de conjuntos de resultados desde procedimientos JDBC" en la página 52
- "Devolución de conjuntos de resultados desde procedimientos de SQLJ" en la página 51

Información relacionada:

- "Sentencia CALL" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia DECLARE CURSOR" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia FETCH" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia ALLOCATE CURSOR" en la publicación *Consulta de SQL, Volumen 2*

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

Recepción de conjuntos de resultados de procedimiento en aplicaciones y rutinas SQLJ

Puede recibir conjuntos de resultados de procedimientos invocados desde una rutina o aplicación SQLJ.

Procedimiento:

Para aceptar conjuntos de resultados de procedimiento desde una rutina o aplicación SQLJ:

1. Abra una conexión de base de datos (utilizando un objeto Connection):
   ```
   Connection con =
   DriverManager.getConnection("jdbc:db2:sample", idusuario, contraseña);
   ```
2. Establezca el contexto por omisión (utilizando un objeto DefaultContext):
   ```
   DefaultContext ctx = new DefaultContext(con);
   DefaultContext.setDefaultContext(ctx);
   ```
3. Establezca el contexto de ejecución (utilizando un objeto ExecutionContext):
   ```
   ExecutionContext execCtx = ctx.getExecutionContext();
   ```
4. Invoque un procedimiento que devuelva conjuntos de resultados. En el ejemplo siguiente, se invoca un procedimiento denominado GET_HIGH_SALARIES, y se pasa una variable de entrada (llamada inSalaryThreshold):
   ```
   #sql {CALL GET_HIGH_SALARIES(:in inSalaryThreshold, :out outErrorCode)};
   ```
5. Declare un objeto ResultSet y utilice el método getNextResultSet() del objeto ExecutionContext a fin de aceptar conjuntos de resultados del procedimiento. Si son varios conjuntos de resultados, coloque la llamada getNextResultSet() en una estructura de bucle. Cada conjunto de resultados devuelto por el procedimiento generará una iteración de bucle. Dentro del bucle, puede captar las filas del conjunto de resultados con su método y luego cerrar el objeto de conjunto de resultados (con el método close() del objeto ResultSet). Por ejemplo:
   ```
   ResultSet rs = null;

   while ((rs = execCtx.getNextResultSet()) != null)
   {
     ResultSetMetaData stmtInfo = rs.getMetaData();
     int numOfColumns = stmtInfo.getColumnCount();
     int r = 0;

     // Las filas del conjunto de resultados se captan y se imprimen en pantalla.
       while (rs.next())
     {
       r++;
       System.out.print("Row: " + r + ": ");
       for (int i=1; i <= numOfColumns; i++)
       {
         System.out.print(rs.getString(i));
         if (i != numOfColumns)
         {
           System.out.print(", ");
         }
       }
       System.out.println();
     }

     rs.close();
   }
   ```

Conceptos relacionados:

- "Conjuntos de resultados de procedimiento" en la página 47

Tareas relacionadas:

- "Devolución de conjuntos de resultados desde procedimientos de SQL y SQL incorporado" en la página 49
- "Devolución de conjuntos de resultados desde procedimientos JDBC" en la página 52
- "Devolución de conjuntos de resultados desde procedimientos de SQLJ" en la página 51

Ejemplos relacionados:

- "SpClient.sqlj -- Call a variety of types of stored procedures from SpServer.sqlj (SQLj)"

Recepción de conjuntos de resultados de procedimiento en aplicaciones y rutinas JDBC

Puede recibir conjuntos de resultados de procedimientos invocados desde una rutina o aplicación JDBC.

Procedimiento:

Para aceptar conjuntos de resultados de procedimiento desde una rutina o aplicación JDBC:

1. Abra una conexión de base de datos (utilizando un objeto Connection):

   ```
   Connection con =
   DriverManager.getConnection("jdbc:db2:sample", idusuario, contraseña);
   ```

2. Prepare la sentencia CALL que invocará a un procedimiento que devuelva conjuntos de resultados (utilizando un objeto CallableStatement). En el ejemplo siguiente, se invoca un procedimiento denominado GET_HIGH_SALARIES. La preparación va seguida de la asignación de una variable de entrada (denominada inSalaryThreshold -- un valor numérico que se debe pasar al procedimiento) al valor del marcador de parámetro de la sentencia anterior. (Un marcador de parámetro se indica con un ″?″.)

   ```
   String query = "CALL GET_HIGH_SALARIES(?)";

   CallableStatement stmt = con.prepareCall(query);
   stmt.setDouble(1, inSalaryThreshold);
   ```

3. Llame al procedimiento:

   ```
   stmt.execute();
   ```

4. Utilice el método getResultSet() del objeto CallableStatement para aceptar el primer conjunto de resultados del procedimiento y capte las filas de los conjuntos de resultados utilizando el método fetchAll():

   ```
   ResultSet rs = stmt.getResultSet();

     // Las filas del conjunto de resultados se captan y se imprimen en pantalla.
       while (rs.next())
   {
     r++;
     System.out.print("Row: " + r + ": ");
     for (int i=1; i <= numOfColumns; i++)
     {
       System.out.print(rs.getString(i));
       if (i != numOfColumns)
       {
   ```

```
        System.out.print(", ");
      }
    }
    System.out.println();
  }
```

5. Para varios conjuntos de resultados, utilice el método getNextResultSet() del objeto CallableStatement a fin de permitir la lectura del siguiente conjunto de resultados. Luego, repita el proceso del paso anterior, en el que el objeto ResultSet acepta el conjunto de resultados actual y capta las filas del conjunto de resultados. Por ejemplo:

```
while (callStmt.getMoreResults())
{
  rs = callStmt.getResultSet()

  ResultSetMetaData stmtInfo = rs.getMetaData();
  int numOfColumns = stmtInfo.getColumnCount();
  int r = 0;

  // Las filas del conjunto de resultados se captan y se imprimen en pantalla.
    while (rs.next())
  {
    r++;
    System.out.print("Row: " + r + ": ");
    for (int i=1; i <= numOfColumns; i++)
    {
      System.out.print(rs.getString(i));
      if (i != numOfColumns)
      {
        System.out.print(", ");
      }
    }
    System.out.println();
  }
}
```

6. Cierre el objeto ResultSet con su método close():

```
    rs.close();
```

Conceptos relacionados:

- "Conjuntos de resultados de procedimiento" en la página 47

Tareas relacionadas:

- "Devolución de conjuntos de resultados desde procedimientos de SQL y SQL incorporado" en la página 49
- "Devolución de conjuntos de resultados desde procedimientos JDBC" en la página 52
- "Devolución de conjuntos de resultados desde procedimientos de SQLJ" en la página 51

Ejemplos relacionados:

- "SpClient.java -- Call a variety of types of stored procedures from SpServer.java (JDBC)"

Manejo de los parámetros en los procedimientos de PROGRAM TYPE MAIN o PROGRAM TYPE SUB

Los procedimientos pueden aceptar parámetros con el estilo de rutinas principales o subrutinas. Esto se determina cuando se registra el procedimiento mediante la sentencia CREATE PROCEDURE.

Los procedimientos C o C++ de PROGRAM TYPE SUB aceptan argumentos del mismo modo que las subrutinas C o C++. Pase los parámetros como punteros. Por ejemplo, la siguiente signatura de procedimiento C acepta parámetros de los tipos INTEGER, SMALLINT y CHAR(3):

```
int storproc (sqlint32 *arg1, sqlint16 *arg2, char *arg3)
```

Los procedimientos Java™ sólo pueden aceptar argumentos como subrutinas. Pase los parámetros IN como simples argumentos. Pase los parámetros OUT e INOUT como matrices con un solo elemento. La siguiente signatura de procedimiento del estilo de parámetros Java acepta un parámetro IN del tipo INTEGER, un parámetro OUT del tipo SMALLINT y un parámetro INOUT del tipo CHAR(3):

```
int storproc (int arg1, short arg2[], String arg[])
```

Para escribir un procedimiento C que acepte argumentos como una función principal de un programa C, especifique PROGRAM TYPE MAIN en la sentencia CREATE PROCEDURE. Debe escribir procedimientos de PROGRAM TYPE MAIN que se ajusten a las especificaciones siguientes:

- El procedimiento acepta parámetros mediante dos argumentos:
  - una variable de contador de parámetros; por ejemplo, *argc*
  - una matriz de punteros para los parámetros; por ejemplo, *char **argv*
- El procedimiento se debe crear como biblioteca compartida

En los procedimientos de PROGRAM TYPE MAIN, DB2® establece el valor del primer elemento de la matriz *argv*, (*argv[0]*), en el nombre del procedimiento. Los elementos restantes de la matriz *argv* corresponden a los parámetros definidos por el PARAMETER STYLE del procedimiento. Por ejemplo, el siguiente procedimiento C intercalado pasa un parámetro IN como *argv[1]* y devuelve dos parámetros OUT como *argv[2]* y *argv[3]*.

La sentencia CREATE PROCEDURE para el ejemplo de PROGRAM TYPE MAIN es la siguiente:

```
CREATE PROCEDURE MAIN_EXAMPLE (IN job CHAR(8),
   OUT salary DOUBLE, OUT errorcode INTEGER)
   DYNAMIC RESULT SETS 0
   LANGUAGE C
   PARAMETER STYLE GENERAL
   NO DBINFO
   FENCED
   READS SQL DATA
   PROGRAM TYPE MAIN
   EXTERNAL NAME 'spserver!mainexample'
```

El código siguiente del procedimiento copia el valor de *argv[1]* en la variable del lenguaje principal CHAR(8) *injob*, luego copia el valor de la variable del lenguaje principal DOUBLE *outsalary* en *argv[2]* y devuelve el SQLCODE como *argv[3]*:

```
SQL_API_RC SQL_API_FN main_example (int argc, char **argv)
{
  EXEC SQL INCLUDE SQLCA;

  EXEC SQL BEGIN DECLARE SECTION;
    char injob[9];
    double outsalary;
  EXEC SQL END DECLARE SECTION;

  /* argv[0] contiene el nombre de procedimiento. */
  /* Los parámetros empiezan en argv[1]         */
  strcpy (injob, (char *)argv[1]);
```

```
    EXEC SQL SELECT AVG(salary)
      INTO :outsalary
      FROM employee
      WHERE job = :injob;

    memcpy ((double *)argv[2], (double *)&outsalary, sizeof(double));

    memcpy ((sqlint32 *)argv[3], (sqlint32 *)&SQLCODE, sizeof(sqlint32));

    return (0);

  } /* fin de la función main_example */
```

Conceptos relacionados:

- "Procedimientos" en la página 13

Información relacionada:

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

Ejemplos relacionados:

- "spcreate.db2 -- How to catalog the stored procedures contained in spserver.sqc (C)"
- "spserver.sqc -- Definition of various types of stored procedures (C)"

Características de las UDF y los métodos

A diferencia de los procedimientos almacenados, las UDF y los métodos se invocan desde sentencias de SQL. Mientras que un procedimiento almacenado se invoca una vez al llamarlo, una función o un método se pueden invocar varias veces desde una sola referencia en una sentencia de SQL. Esta diferencia en la implementación requiere características especiales. Los apartados siguientes describen las áreas reutilizables, las cuales se pueden emplear para conservar la información de estado entre las invocaciones, y el modelo de proceso de las UDF y los métodos que se registran con la opción FINAL CALL.

Áreas reutilizables para UDF y métodos

Un *área reutilizable* permite que una función definida por el usuario o un método guarden su estado entre una invocación y la siguiente. Por ejemplo, a continuación se identifican dos situaciones en las que guardar el estado entre las invocaciones es beneficioso:

1. Las funciones o los métodos que, para ser correctos, dependen de que se guarde el estado.

   Un ejemplo de función o método de este tipo es una simple función de contador que devuelve un '1' la primera vez que recibe llamada, e incrementa el resultado en uno en cada llamada sucesiva. En algunas circunstancias, este tipo de función se podría utilizar para numerar las filas del resultado de una sentencia SELECT:

   ```
   SELECT counter(), a, b+c, ...
   FROM tablex
   WHERE ...
   ```

   La función necesita un lugar en el que almacenar el valor actual del contador entre las invocaciones, donde se garantice que el valor será el mismo para la siguiente invocación. Luego, en cada invocación se puede incrementar el valor y devolverlo como resultado de la función.

Este tipo de rutina es NOT DETERMINISTIC. Su salida no depende únicamente de los valores de sus argumentos de SQL.

2. Las funciones o los métodos en que se puede mejorar el rendimiento mediante la posibilidad de realizar algunas acciones de inicialización.

   Un ejemplo de función o método de este tipo, que puede formar parte de una aplicación de documento, es una función *match* (de coincidencia), que devuelve 'Y' si un documento determinado contiene una serie indicada, y 'N' en caso contrario:

   ```
   SELECT docid, doctitle, docauthor
   FROM docs
   WHERE match('myocardial infarction', docid) = 'Y'
   ```

   Esta sentencia devuelve todos los documentos que contienen el valor de serie de texto concreto representado por el primer argumento. Lo que *match* desearía hacer es:

   - Sólo la primera vez.

     Recuperar una lista de todos los ID de documento que contengan la serie 'myocardial infarction' desde la aplicación de documento, que se mantiene fuera de DB2®. Esta recuperación es un proceso costoso, por lo que la función desearía hacerlo una sola vez, y guardar la lista en algún lugar para tenerla a mano en las llamadas posteriores.

   - En cada llamada.

     Utilizar la lista de los ID de documento guardada durante la primera llamada, para ver si el ID de documento que se pasa como segundo argumento está incluido en la lista.

   Este tipo de rutina es DETERMINISTIC. Su respuesta sólo depende de los valores de los argumentos de entrada. Lo que se muestra aquí es una función cuyo rendimiento, y no su corrección, depende de la posibilidad de guardar información entre una llamada y la siguiente.

Ambas necesidades se satisfacen con la posibilidad de especificar un área reutilizable (SCRATCHPAD) en la sentencia CREATE:

```
CREATE FUNCTION counter()
  RETURNS int ... SCRATCHPAD;

CREATE FUNCTION match(varchar(200), char(15))
RETURNS char(1) ... SCRATCHPAD 10000;
```

La palabra clave SCRATCHPAD indica a DB2 que asigne y mantenga un área reutilizable para una rutina. El tamaño por omisión de un área reutilizable es de 100 bytes, pero el usuario puede determinar el tamaño (en bytes) correspondiente a un área reutilizable. El ejemplo de *match* presenta 10000 bytes de longitud. DB2 inicializa el área reutilizable con ceros binarios antes de la primera invocación. Si el área reutilizable se define para una función de tabla, y si la función de tabla también se define con NO FINAL CALL (valor por omisión), DB2 renueva el área reutilizable antes de cada llamada a OPEN. Si se especifica la opción de función de tabla FINAL CALL, DB2 no examinará ni cambiará el contenido del área reutilizable después de su inicialización. Para las funciones escalares definidas con áreas reutilizables, DB2 tampoco examina ni cambia el contenido del área después de su inicialización. En cada invocación se pasa a la rutina un puntero al área reutilizable, y DB2 conserva la información del estado de la rutina en el área reutilizable.

Así, para el ejemplo de *counter*, el último valor devuelto se puede conservar en el área reutilizable. Y, en el ejemplo de *match*, se puede conservar en el área reutilizable la lista de documentos, en caso de que el área reutilizable sea

suficientemente grande; de lo contrario, se puede asignar memoria para la lista y conservar la dirección de la memoria adquirida en el área reutilizable. Las áreas reutilizables pueden tener una longitud variable: la longitud se define en la sentencia CREATE para la rutina.

El área reutilizable únicamente se aplica a la referencia individual a la rutina en la sentencia. Si en una sentencia existen varias referencias a una rutina, cada referencia tiene su propia área reutilizable, por lo que estas áreas no se pueden emplear para realizar comunicaciones entre referencias. El área reutilizable sólo se aplica a un único agente de DB2 (un agente es una entidad de DB2 que realiza el proceso de todos los aspectos de una sentencia). No existe un ″área reutilizable global″ para coordinar el compartimiento de la información de las áreas reutilizables entre los agentes. Esto es importante, en concreto, para aquellas situaciones en que DB2 establece varios agentes para procesar una sentencia (ya sea en una base de datos de una sola partición o de varias). En estos casos, aunque una sentencia puede contener una sola referencia a una rutina, pueden existir varios agentes que realicen el trabajo y cada uno de ellos tendrá su propia área reutilizable. En una base de datos de varias particiones, donde una sentencia que hace referencia a una UDF ha de procesar datos en varias particiones e invocar la UDF en cada partición, el área reutilizable sólo se aplica a una partición. Como consecuencia, existe un área reutilizable en cada partición en que se ejecuta la UDF.

Si la ejecución correcta de una función depende de que haya una sola área reutilizable por cada referencia a la función, registre la función como DISALLOW PARALLEL. Esto causará que la función se ejecute en una única partición y, de esta manera, se garantizará que exista una única área reutilizable por cada referencia a la función.

Puesto que es reconocido que una UDF o un método pueden requerir recursos del sistema, la UDF o el método se pueden definir con la palabra clave FINAL CALL. Esta palabra clave indica a DB2 que llame a la UDF o al método durante el proceso de fin de sentencia, para que la UDF o el método puedan liberar sus recursos del sistema. Es vital que una rutina libere cualquier recurso que adquiera; incluso una pequeña fuga se puede convertir en una gran fuga en un entorno en que la sentencia se invoque repetidamente, y una gran fuga puede provocar una detención de DB2 por anomalía grave.

Puesto que el área reutilizable tiene un tamaño fijo, es posible que la UDF o el método incluyan una asignación de memoria y por ello puedan utilizar la llamada final para liberar la memoria. Por ejemplo, la función *match* anterior no puede predecir cuántos documentos coincidirán con la serie de texto indicada. Por lo tanto, la siguiente resultará una definición mejor para *match*:

```
CREATE FUNCTION match(varchar(200), char(15))
RETURNS char(1) ... SCRATCHPAD 10000 FINAL CALL;
```

Para las UDF o los métodos que emplean un área reutilizable y están referidos en una subconsulta, DB2 puede efectuar una llamada final, si la UDF o el método se especifican así, y renovar el área reutilizable entre invocaciones de la subconsulta. El usuario se puede proteger frente a esta posibilidad, si las UDF o los métodos se utilizan alguna vez en subconsultas, definiendo la UDF o el método con FINAL CALL y utilizando el argumento de tipo de llamada o bien comprobando siempre el estado de *cero binario* del área reutilizable.

Si no especifica FINAL CALL, observe que la UDF o el método recibirán una llamada del tipo FIRST. Se puede utilizar esta posibilidad para adquirir e inicializar algún recurso persistente.

Conceptos relacionados:

- "Áreas reutilizables en sistemas operativos de 32 bits y 64 bits" en la página 61
- "Modelo de proceso de los métodos y las funciones escalares" en la página 62

Información relacionada:

- "Sentencia CREATE FUNCTION" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE TYPE (Estructurado)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE METHOD" en la publicación *Consulta de SQL, Volumen 2*

Áreas reutilizables en sistemas operativos de 32 bits y 64 bits

Para hacer que el código de la UDF o del método se pueda transportar entre sistemas operativos de 32 bits y 64 bits, debe ir con precaución en la manera de crear y utilizar las áreas reutilizables que contengan valores de 64 bits. Es recomendable no declarar una variable de longitud explícita para una estructura de área reutilizable que contenga uno o más valores de 64 bits, tales como los punteros de 64 bits o las variables BIGINT `sqlint64`.

Un área reutilizable se pasa en forma de LOB, que tiene la estructura siguiente:

```
struct lob
{
   sqlint32 length;
   char data[100];
}
```

Cuando se define su propia estructura para el área reutilizable, una rutina tiene dos opciones:

1. Redefinir el LOB de área reutilizable entero, en cuyo caso es necesario incluir un campo de longitud explícito. Por ejemplo:

   ```
   struct spadlob
   {
     sqlint32 lob_length;
     sqlint32 int_var;
     sqlint64 bigint_var;
   };
   void SQL_API_FN routine( ..., struct spadlob* scratchpad, ... )
   {
     /* Utilizar área reutilizable */
   }
   ```

2. Redefinir solamente la porción de datos del LOB de área reutilizable, en cuyo caso no se necesita ningún campo de longitud.

   ```
   struct spaddata
   {
     sqlint32 int_var;
     sqlint64 bigint_var;
   };
   void SQL_API_FN routine( ..., struct lob* lob_spad, ... )
   {
     struct spaddata* scratchpad = (struct spaddata*)lob_spad-->data;
     /* Utilizar área reutilizable */
   }
   ```

Puesto que la aplicación no puede cambiar el valor del campo de longitud del LOB de área reutilizable, el hecho de codificar la rutina tal como se muestra en el primer ejemplo no brinda ningún beneficio significativo. El segundo ejemplo también se puede transportar entre sistemas de distintos tamaños de palabra, por lo que representa la manera preferible de escribir la rutina.

Conceptos relacionados:

- "Áreas reutilizables para UDF y métodos" en la página 58
- "Funciones escalares definidas por el usuario" en la página 15
- "Funciones escalares definidas por el usuario" en la página 17

Tareas relacionadas:

- "Invocación de rutinas de 32 bits en un servidor de bases de datos de 64 bits" en la página 213

Modelo de proceso de los métodos y las funciones escalares

El modelo de proceso para los métodos y las UDF escalares que se definen con la especificación FINAL CALL es el siguiente:

Llamada FIRST
Éste es un caso especial de la llamada NORMAL, identificado como FIRST para permitir que la función realice cualquier proceso inicial. Se evalúan los argumentos y se pasan a la función. Normalmente, la función devolverá un valor en esta llamada, pero puede devolver un error, en cuyo caso no se realiza ninguna llamada NORMAL ni FINAL. Si se devuelve un error en una llamada FIRST, lo debe borrar el método o la UDF antes de volver, puesto que no se realizará ninguna llamada FINAL.

Llamada NORMAL
Son las llamadas a la función que van de la segunda a la última, según indiquen los datos y la lógica de la sentencia. Se espera que la función devuelva un valor con cada llamada NORMAL después de que se evalúen y pasen los argumentos. Si una llamada NORMAL devuelve un error, no se realiza ninguna otra llamada NORMAL pero se realiza la llamada FINAL.

Llamada FINAL
Ésta es una llamada especial, que se realiza en el proceso de fin de sentencia (o en el cierre (CLOSE) de un cursor), siempre y cuando la llamada FIRST sea satisfactoria. En una llamada FINAL no se pasa ningún valor de argumento. Esta llamada se lleva a cabo para que la función pueda borrar los recursos. La función no devuelve un valor en esta llamada, pero puede devolver un error.

Si se trata de métodos o UDF escalares que no se han definido con FINAL CALL, sólo se realizan llamadas NORMAL a la función, la cual habitualmente devuelve un valor para cada llamada. Si una llamada NORMAL devuelve un error, o si la sentencia encuentra otro error, no se realiza ninguna otra llamada a la función.

Nota: Este modelo describe el proceso de errores corriente para los métodos y las UDF escalares. En el caso de una anomalía del sistema o un problema de comunicación, no se puede efectuar una llamada indicada por el modelo de proceso de errores. Por ejemplo, para una UDF FENCED, si el proceso protegido `db2udf` termina de algún modo de forma prematura, DB2 no puede efectuar las llamadas indicadas.

Conceptos relacionados:

- "Funciones escalares definidas por el usuario" en la página 15
- "Métodos" en la página 18

Funciones de tabla definidas por el usuario

Además de devolver valores escalares, también se pueden desarrollar UDF que devuelvan tablas. Los apartados siguientes describen las funciones de tabla definidas por el usuario y el modelo de proceso correspondiente a las UDF de tabla registradas con la opción FINAL CALL.

Funciones de tabla definidas por el usuario

Una función de tabla definida por el usuario entrega una tabla al SQL en el que se le hace referencia. Una referencia a una UDF de tabla sólo es válida en una cláusula FROM de una sentencia SELECT. Cuando utilice funciones de tabla, tenga en cuenta lo siguiente:

- Aunque una función de tabla entrega una tabla, la interfaz física entre DB2® y la UDF es de una fila cada vez. Hay cinco tipos de llamadas que se pueden realizar a una función de tabla: OPEN, FETCH, CLOSE, FIRST y FINAL. La existencia de las llamadas FIRST y FINAL depende de cómo se defina la UDF. Para distinguir estas llamadas, se utiliza el mismo mecanismo de *tipo-llamada* que se usa para las funciones escalares.
- No se tienen que devolver todas las columnas de resultado definidas en la cláusula RETURNS de la sentencia CREATE FUNCTION para la función de tabla. La palabra clave DBINFO de CREATE FUNCTION y el argumento *dbinfo* correspondiente permiten una optimización consistente en que sólo haya que devolver las columnas necesarias para una referencia a una función de tabla determinada.
- Los valores de columna individuales devueltos se ajustan al formato de los valores devueltos por las funciones escalares.
- La sentencia CREATE FUNCTION para una función de tabla tiene la especificación CARDINALITY. Esta especificación permite que el creador informe al optimizador de DB2 del tamaño aproximado del resultado, de forma que el optimizador pueda tomar decisiones mejores cuando se haga referencia a la función.

  Independientemente de lo que se haya especificado como CARDINALITY de una función de tabla, tenga cuidado respecto a la escritura de una función con una cardinalidad infinita, es decir, una función que siempre devuelva una fila en una llamada FETCH. Existen muchas situaciones en las que DB2 espera la condición de fin de tabla como catalizador dentro del proceso de la consulta. La utilización de GROUP BY u ORDER BY son ejemplos de este caso. DB2 no puede formar los grupos de agregación hasta que se llega al fin de tabla, ni puede realizar ninguna clasificación sin tener todos los datos. Por lo tanto, una función de tabla que nunca devuelva la condición de fin de tabla (valor '02000' de estado de SQL) puede ocasionar un bucle infinito del proceso si se utiliza con una cláusula GROUP BY u ORDER BY.

Información relacionada:

- "Sentencia CREATE FUNCTION" en la publicación *Consulta de SQL, Volumen 2*
- "Sintaxis para pasar argumentos a rutinas escritas en C/C++, OLE o COBOL" en la página 97

Modelo de proceso de las funciones de tabla

El modelo de proceso para las UDF de tabla que se definen con la especificación FINAL CALL es el siguiente:

Llamada FIRST
Esta llamada se realiza antes de la primera llamada OPEN y su objetivo consiste en permitir que la función realice cualquier proceso inicial. Antes de esta llamada, el área reutilizable se borra. Se evalúan los argumentos y se pasan a la función. La función no devuelve una fila. Si la función devuelve un error, no se realiza ninguna otra llamada a la misma.

Llamada OPEN
Esta llamada se realiza para permitir que la función realice un proceso especial de apertura (OPEN) específico de la exploración. El área reutilizable (si existe) no se borra antes de la llamada. Se evalúan los argumentos y se pasan. La función no devuelve una fila en una llamada OPEN. Si la función devuelve un error de la llamada OPEN, no se realizará ninguna llamada FETCH ni CLOSE, pero sí se realizará la llamada FINAL al final de la sentencia.

Llamada FETCH
Se siguen produciendo llamadas FETCH hasta que la función devuelve un valor de SQLSTATE que significa fin-de-tabla. Es en estas llamadas donde la UDF desarrolla y devuelve una fila de datos. Se pueden pasar valores de argumentos a la función, pero éstos apuntan a los mismos valores que se han pasado al ejecutar OPEN. Por lo tanto, es posible que los valores de argumentos no sean actuales y no se debe confiar en ellos. Si tiene necesidad de mantener valores actuales entre las invocaciones de una función de tabla, utilice un área reutilizable. La función puede devolver un error de una llamada FETCH, y la llamada CLOSE se seguirá produciendo.

Llamada CLOSE
Se realiza esta llamada cuando finaliza la exploración o sentencia, siempre que la llamada OPEN haya resultado satisfactoria. Los posibles valores de argumentos no serán actuales. La función puede devolver un error.

Llamada FINAL
Se realiza la llamada FINAL al final de la sentencia, siempre que la llamada FIRST haya resultado satisfactoria. Esta llamada se lleva a cabo para que la función pueda borrar los recursos. La función no devuelve un valor en esta llamada, pero puede devolver un error.

Para las UDF de tabla no definidas con FINAL CALL, sólo se realizan las llamadas OPEN, FETCH y CLOSE a la función. Antes de cada llamada OPEN, se borra el área reutilizable (si existe).

La diferencia entre las UDF de tabla definidas con FINAL CALL y las definidas con NO FINAL CALL se puede observar examinando un escenario que implique una unión o una subconsulta, en que el acceso de la función de tabla es el acceso ″interior″. Por ejemplo, en una sentencia como la siguiente:

```
SELECT x,y,z,... FROM table_1 as A,
  TABLE(table_func_1(A.col1,...)) as B
  WHERE...
```

En este caso, el optimizador abrirá una exploración de table_func_1 para cada fila de table_1. Esto es debido a que el valor de la col1 de table_1, que se pasa a table_func_1, se utiliza para definir la exploración de la función de tabla.

Para las UDF de tabla con NO FINAL CALL, la secuencia de llamadas OPEN, FETCH, FETCH, ..., CLOSE se repite para cada fila de table_1. Observe que cada llamada OPEN obtendrá un área reutilizable limpia. Puesto que la función de tabla no sabe, al final de cada exploración, si se producirán más exploraciones, la tiene

que limpiar completamente durante el proceso de cierre (CLOSE). Esto puede resultar ineficaz si existe un proceso significativo de apertura de una sola vez que se debe repetir.

Las UDF de tabla con FINAL CALL proporcionan una llamada FIRST de una sola vez y una llamada FINAL de una sola vez. Estas llamadas se utilizan para amortizar los costes de inicialización y terminación a lo largo de todas las exploraciones de la función de tabla. Al igual que anteriormente, las llamadas OPEN, FETCH, FETCH, ..., CLOSE se realizan para cada fila de la tabla exterior pero, dado que la función de tabla sabe que obtendrá una llamada FINAL, no es necesario que efectúe una limpieza completa en la llamada CLOSE (ni que la reasigne en la OPEN posterior). Observe también que el área reutilizable no se borra entre las exploraciones, en gran parte porque los recursos de la función de tabla se repartirán a lo largo de las exploraciones.

A expensas de gestionar dos tipos de llamadas adicionales, la UDF de tabla puede alcanzar una eficacia mayor en estos escenarios de unión y subconsulta. La decisión de si se debe definir la función de tabla como FINAL CALL depende del modo en que se pretenda utilizar.

Conceptos relacionados:

- "Modelo de ejecución de funciones de tabla para Java" en la página 65
- "Funciones escalares definidas por el usuario" en la página 17

Información relacionada:

- "Sentencia CREATE FUNCTION (Tabla externa OLE DB)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE FUNCTION (Escalar de SQL, tabla o fila)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE FUNCTION (Tabla externa)" en la publicación *Consulta de SQL, Volumen 2*

Modelo de ejecución de funciones de tabla para Java

Para las funciones de tabla escritas en Java™ que utilizan PARAMETER STYLE DB2GENERAL, es importante comprender lo que sucede en cada punto del proceso de DB2® de una sentencia determinada. La tabla siguiente detalla esta información para una función de tabla habitual. Se abarcan los casos de NO FINAL CALL y de FINAL CALL, suponiendo en ambos casos SCRATCHPAD.

Punto en el tiempo de exploración	NO FINAL CALL LANGUAGE JAVA SCRATCHPAD	FINAL CALL LANGUAGE JAVA SCRATCHPAD
Antes de la primera OPEN para la función de tabla	No hay llamadas.	• Se llama al constructor de clases (por medio de la nueva área reutilizable). Se llama al método de UDF con la primera (FIRST) llamada. • El constructor inicializa las variables de clase y área reutilizable. El método conecta con el servidor de Web.

Punto en el tiempo de exploración	NO FINAL CALL LANGUAGE JAVA SCRATCHPAD	FINAL CALL LANGUAGE JAVA SCRATCHPAD
En cada OPEN de la función de tabla	• Se llama al constructor de clases (por medio de la nueva área reutilizable). Se llama al método de UDF con la llamada OPEN. • El constructor inicializa las variables de clase y área reutilizable. El método conecta con el servidor de Web y abre la exploración de los datos de la Web.	• Se abre el método de UDF con la llamada OPEN. • El método abre la exploración de los datos de la Web que desee. (Puede ser capaz de evitar que se tenga que volver a abrir después de una reposición de CLOSE, según lo que se guarde en el área reutilizable.)
En cada FETCH para una nueva fila de datos de la función de tabla	• Se llama al método de UDF con la llamada FETCH. • El método capta y devuelve la siguiente fila de datos, o bien EOT.	• Se llama al método de UDF con la llamada FETCH. • El método capta y devuelve la nueva fila de datos, o bien EOT.
En cada CLOSE de la función de tabla	• Se llama al método de UDF con la llamada CLOSE. Método `close()`, si existe para la clase. • El método cierra su exploración de la Web y se desconecta del servidor de Web. No es necesario que `close()` haga nada.	• Se llama al método de UDF con la llamada CLOSE. • El método puede reposicionarse al principio de la exploración o cerrarla. Puede guardar cualquier estado en el área reutilizable, el cual persistirá.
Después de la última CLOSE de la función de tabla	No hay llamadas.	• Se llama al método de UDF con la llamada FINAL. Se llama al método `close()`, si existe para la clase. • El método se desconecta del servidor de Web. No es necesario que el método `close()` haga nada.

Notas:

1. El término ″método de UDF″ hace referencia al método de clase de Java que implementa la UDF. Se trata del método identificado en la cláusula EXTERNAL NAME de la sentencia CREATE FUNCTION.
2. Para las funciones de tabla que tengan especificado NO SCRATCHPAD, las llamadas al método de UDF son las indicadas en esta tabla, pero, puesto que el usuario no solicita ninguna continuidad por medio de un área reutilizable, DB2 hará que se cree una instancia de un nuevo objeto antes de cada llamada, llamando al constructor de clases. No está claro que las funciones de tabla con NO SCRATCHPAD (y, por lo tanto, sin continuidad) puedan realizar acciones útiles, pero se soportan.

Conceptos relacionados:

- “Rutinas DB2GENERAL” en la página 357
- “Rutinas Java” en la página 181
- “Modelo de proceso de las funciones de tabla” en la página 63

Información relacionada:

- “Sentencia CREATE FUNCTION (Tabla externa)” en la publicación *Consulta de SQL, Volumen 2*

Capítulo 3. Rutinas de SQL

SQL Procedural Language (SQL PL) en DB2 . . . 67
Sentencias CREATE para las rutinas de SQL . . . 68
Niveles de acceso de SQL en rutinas de SQL . . . 69
SQL dinámico en las rutinas de SQL 69
Procedimientos de SQL/ SQL PL 71
Consideraciones de diseño para los procedimientos de SQL 71
Creación de procedimientos de SQL desde la línea de mandatos 72
Parámetros de los procedimientos de SQL . . . 74
Variables en los procedimientos de SQL (sentencias DECLARE, DEFAULT, SET) 75
Bloques compuestos y ámbito de las variables en los procedimientos de SQL 76
Devolución de mensajes de error desde procedimientos de SQL 77
Manejadores de condiciones en los procedimientos de SQL 77
Manejadores de condiciones en los procedimientos de SQL 78
Declaraciones de manejadores de condiciones 78
Sentencias SIGNAL y RESIGNAL en manejadores de condiciones 81
Variables SQLCODE y SQLSTATE en procedimientos de SQL 81
Mejora del rendimiento de los procedimientos de SQL 82
Funciones de tabla de SQL 88
Funciones de tabla de SQL que modifican datos de SQL 88
Auditoría mediante funciones de tabla de SQL 90

Las rutinas de SQL se crean ejecutando la sentencia CREATE adecuada para el tipo de rutina, en la que también se especifica el cuerpo de la rutina, que, en el caso de una rutina de SQL, debe estar compuesto totalmente por sentencias de SQL o SQL PL. Puede utilizar el Centro de Desarrollo de IBM DB2 como ayuda en la creación, depuración y ejecución de procedimientos de SQL o bien puede crearlos mediante el procesador de línea de mandatos de DB2.

SQL Procedural Language (SQL PL) en DB2

Antes de describir el uso del SQL PL en los procedimientos y las funciones, es importante revisar primero la terminología básica y algunos conceptos relacionados con el SQL de procedimientos en DB2. Las construcciones del SQL de procedimientos, tales como variables escalares, sentencias IF y bucles WHILE, se presentaron en DB2 con el release de DB2 Versión 7. Estas construcciones se han ampliado en el conjunto de sentencias de SQL que componen el SQL Procedural Language (SQL PL) al que se hace referencia ahora.

SQL PL:

El SQL PL es, en realidad, un subconjunto del SQL que proporciona construcciones de procedimiento que se pueden utilizar para implementar la lógica alrededor de las sentencias de SQL tradicionales. El SQL PL es un lenguaje de programación de alto nivel con una sintaxis sencilla y sentencias habituales de control de programación, que incluyen las sentencias IF, ELSE, WHILE, FOR, ITERATE y GOTO, así como otras sentencias.

Procedimientos de SQL PL y de SQL:

Los procedimientos de SQL PL pueden contener parámetros, variables, sentencias de asignación, sentencias de control de SQL PL y sentencias de SQL compuestas. Los procedimientos de SQL PL también dan soporte a un potente mecanismo de manejo de errores y condiciones, a las llamadas anidadas y repetitivas y a la devolución de varios conjuntos de resultados al llamante o a la aplicación cliente. Para conocer el conjunto completo de elementos del lenguaje soportado en los procedimientos de SQL PL, vea la sentencia CREATE PROCEDURE (SQL) en la Consulta de SQL.

Funciones, activadores y sentencias dinámicas compuestas de SQL PL en línea y de SQL:

A partir de DB2 Versión 7.2, está soportado un subconjunto de SQL PL en los cuerpos de los activadores y las funciones de SQL. Este subconjunto de SQL PL se conoce como SQL PL en línea. La palabra en línea hace resaltar una diferencia importante entre el SQL PL en línea y el lenguaje SQL PL completo. Mientras que un procedimiento de SQL PL se implementa compilando estáticamente sus consultas de SQL individuales en secciones de un paquete, una función de SQL PL en línea se implementa, como su nombre sugiere, colocando en línea el cuerpo de la función en la consulta que la utiliza. De esta diferencia, surgen algunas consideraciones relativas al rendimiento y se deben tener en cuenta cuando planifique si ha de implementar la lógica de procedimiento en SQL PL en un procedimiento o utilizar el SQL PL en línea.

Una sentencia dinámica compuesta es una sentencia que, en realidad, le permite agrupar varias sentencias de SQL en un bloque lógico atómico pequeño, en el cual puede declarar variables y elementos para el manejo de condiciones. DB2 compila estas sentencias como una sola sentencia de SQL, y pueden contener elementos de SQL PL. Dentro de una sentencia dinámica compuesta, es posible incluir el subconjunto de SQL PL conocido como SQL PL en línea y tan sólo un pequeño conjunto de sentencias de SQL básicas. Las sentencias dinámicas compuestas son útiles para crear scripts reducidos que realicen pequeñas unidades de trabajo lógico con un mínimo flujo de control, pero con un flujo de datos significativo. Si le interesa una lógica más compleja que requiera parámetros, pase de conjuntos de resultados u otros elementos de procedimiento más avanzados, pueden convenirle más los procedimientos y las funciones de SQL.

Para obtener la lista completa de las sentencias de SQL PL que están soportadas en los procedimientos de SQL PL, funciones de SQL y sentencias dinámicas compuestas, incluidos los activadores, vea las sentencias CREATE referentes a cada tipo de rutina en la Consulta de SQL.

Conceptos relacionados:

- "Rutinas en el desarrollo de aplicaciones" en la página 3
- "Rutinas definidas por el usuario" en la página 10
- "Sentencias CREATE para las rutinas de SQL" en la página 68

Tareas relacionadas:

- "Creación de procedimientos de SQL desde la línea de mandatos" en la página 72

Sentencias CREATE para las rutinas de SQL

Las rutinas de SQL se crean ejecutando la sentencia CREATE adecuada para el tipo de rutina, en la que también se especifica el cuerpo de la rutina, que, en el caso de una rutina de SQL, debe estar compuesto únicamente por sentencias de SQL o SQL PL. Puede utilizar el Centro de Desarrollo de IBM DB2 como ayuda en la creación, depuración y ejecución de procedimientos de SQL. Los procedimientos, funciones y métodos de SQL también se pueden crear mediante el procesador de línea de mandatos de DB2.

Los procedimientos, funciones y métodos de SQL tienen, cada cual, sus sentencias CREATE respectivas. La sintaxis de estas sentencias es diferente; no obstante, existen algunos elementos comunes para ellas. En cada una, debe especificar el

nombre de la rutina, y los parámetros si debe haber alguno, así como un tipo de retorno. Además, puede especificar palabras clave adicionales que proporcionen información a DB2 sobre la lógica incluida en la rutina. DB2 utiliza el prototipo de rutina y las palabras clave adicionales para identificar la rutina durante la invocación y para ejecutarla con el soporte de las características necesarias y el mejor rendimiento posible.

Si desea información específica sobre cómo crear procedimientos de SQL en el Centro de Desarrollo de DB2 o desde el Procesador de línea de mandatos, o sobre cómo crear funciones y métodos, consulte los temas relacionados.

Niveles de acceso de SQL en rutinas de SQL

Un nivel de acceso de SQL es una cláusula especificada en una sentencia CREATE para una rutina que indica el nivel de acceso de SQL utilizado en la rutina. Esta cláusula se emplea para proporcionar información al gestor de bases de datos sobre la sentencia a fin de que el gestor de bases de datos pueda ejecutar dicha sentencia de forma segura y con el mejor rendimiento posible.

Por omisión, los procedimientos de SQL se crean con el nivel de acceso de SQL MODIFIES SQL DATA. Éste se puede cambiar por un nivel inferior de acceso, como READS SQL DATA o CONTAINS SQL, si la sentencia de SQL no modifica datos de tabla dentro del procedimiento. Para ello, especifique la cláusula del nivel de acceso de SQL adecuado en la sentencia CREATE del procedimiento. Se consigue un rendimiento óptimo de las rutinas cuando se especifica la cláusula de acceso de SQL más restrictiva que sea válida en la sentencia CREATE.

Por omisión, todas las UDF (funciones de tabla, funciones escalares, métodos) se crean con el nivel de acceso de SQL READS SQL DATA. El nivel de acceso de SQL se puede definir o modificar con CONTAINS SQL para las UDF incorporadas al SQL cuando no se leen datos dentro de la UDF. El nivel de acceso de SQL de las funciones de tabla incorporadas al SQL se puede definir o modificar con MODIFIES SQL DATA porque, en su cuerpo, están soportadas sentencias de SQL que modifican tablas.

Conceptos relacionados:

- "Funciones de tabla de SQL que modifican datos de SQL" en la página 88

Información relacionada:

- "Sentencias de SQL soportadas" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente*

SQL dinámico en las rutinas de SQL

Las rutinas de SQL, al igual que las rutinas externas, pueden emitir sentencia de SQL dinámico. Si la sentencia de SQL dinámico no incluye marcadores de parámetros y la piensa ejecutar una sola vez, utilice la sentencia EXECUTE IMMEDIATE.

Si la sentencia de SQL dinámico contiene marcadores de parámetros, debe utilizar las sentencias PREPARE y EXECUTE. Si tiene la intención de ejecutar varias veces una sentencia de SQL dinámico, le puede resultar más eficaz emitir una sola sentencia PREPARE y emitir la sentencia EXECUTE varias veces, en lugar de emitir la sentencia EXECUTE IMMEDIATE cada vez.

Para utilizar las sentencias PREPARE y EXECUTE con la finalidad de emitir SQL dinámico en la rutina de SQL, debe incluir las sentencias siguientes en el cuerpo de la rutina de SQL:

1. Declare una variable de tipo VARCHAR, que sea suficientemente grande para contener la sentencia de SQL dinámico, mediante una sentencia DECLARE.
2. Asigne una serie de sentencia a la variable mediante una sentencia SET. No puede incluir las variables directamente en la serie de sentencia. En cambio, debe utilizar el símbolo de interrogación ('?') como marcador de parámetros para las variables que utilice en la sentencia.
3. Cree una sentencia preparada a partir de la serie de sentencia, mediante una sentencia PREPARE.
4. Ejecute la sentencia preparada mediante una sentencia EXECUTE. Si la serie de sentencia incluye marcadores de parámetros de entrada, utilice la cláusula USING para sustituirla por el valor de una variable. Si la sentencia incluye marcadores de parámetros de salida, utilice la cláusula INTO para especificar las variables que recibirán la salida.

Nota: Los nombres de sentencia definidos en las sentencias PREPARE para rutinas de SQL se tratan como variables con ámbito. Una vez que la rutina de SQL sale del ámbito en que se ha definido el nombre de sentencia, DB2® ya no puede acceder a dicho nombre. Dentro de cualquier sentencia compuesta, no se pueden emitir dos sentencias PREPARE que utilicen el mismo nombre de sentencia.

El ejemplo siguiente muestra un procedimiento de SQL que incluye sentencias de SQL dinámico:

El procedimiento de SQL recibe un número de departamento, (*deptNumber*), como parámetro de entrada. En el procedimiento de SQL, se construyen, preparan y ejecutan tres series de sentencia. La primera serie de sentencia ejecuta una sentencia DROP para asegurarse de que la tabla que se va a crear aún no existe. Esta tabla se llama DEPT_*deptno*_T, siendo *deptno* el valor del parámetro de entrada *deptNumber*. Una cláusula CONTINUE HANDLER asegura que el procedimiento de SQL continuará si detecta SQLSTATE 42704 (“nombre de objeto indefinido”), que DB2 devuelve desde la sentencia DROP en caso de que no exista la tabla. La segunda serie de sentencia emite una sentencia CREATE para crear DEPT_*deptno*_T. La tercera serie de sentencia inserta filas para empleados del departamento *deptno* en DEPT_*deptno*_T. La tercera serie de sentencia contiene un marcador de parámetros que representa a *deptNumber*. Cuando se ejecuta la sentencia preparada, el parámetro *deptNumber* sustituye al marcador de parámetros.

```
CREATE PROCEDURE create_dept_table
(IN deptNumber VARCHAR(3), OUT table_name VARCHAR(30))
LANGUAGE SQL
  BEGIN
    DECLARE stmt VARCHAR(1000);

    -- continue if sqlstate 42704 ('nombre de objeto indefinido')
    DECLARE CONTINUE HANDLER FOR SQLSTATE '42704'
      SET stmt = '';
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
      SET table_name = 'PROCEDURE_FAILED';

    SET table_name = 'DEPT_'||deptNumber||'_T';
    SET stmt = 'DROP TABLE '||table_name;
    PREPARE s1 FROM stmt;
    EXECUTE s1;
```

```
    SET stmt = 'CREATE TABLE '||table_name||
     '( empno CHAR(6) NOT NULL, '||
     'firstnme VARCHAR(12) NOT NULL, '||
     'midinit CHAR(1) NOT NULL, '||
     'lastname VARCHAR(15) NOT NULL, '||
     'salary DECIMAL(9,2))';
    PREPARE s2 FROM STMT;
    EXECUTE s2;
    SET stmt = 'INSERT INTO '||table_name || ' ' ||
     'SELECT empno, firstnme, midinit, lastname, salary '||
     'FROM employee '||
     'WHERE workdept = ?';
   PREPARE s3 FROM stmt;
   EXECUTE s3 USING deptNumber;
END
```

Conceptos relacionados:

- "Sentencias de soporte de SQL dinámico" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente*

Información relacionada:

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

Procedimientos de SQL/ SQL PL

Consideraciones de diseño para los procedimientos de SQL

Cuando tome en consideración diseñar un procedimiento de SQL, debe tener en cuenta lo siguiente:

¿Es un procedimiento lo que necesito realmente?
Consulte la comparación de los tipos de rutinas para saber cómo se utilizan los procedimientos y otros tipos de rutinas y para comparar las características y restricciones de cada tipo.

¿Las sentencias de SQL que tengo que ejecutar están soportadas en los procedimientos de SQL?
Para comprobarlo, vea la Consulta de SQL. Si las sentencias que requiere no están soportadas en un procedimiento de SQL, tome en consideración implementar la lógica necesaria en un procedimiento externo, función de SQL o aplicación.

¿Bastaría con sólo una sentencia dinámica compuesta para cumplir mis requisitos?
A veces, sucede que, para las partes muy pequeñas y simples de la lógica de procedimiento, basta con una sentencia de SQL compuesta. Las sentencias compuestas le permiten agrupar varias sentencias como una unidad para ejecutarlas juntas, y pueden incluir algunos elementos de lenguaje SQL PL. Las sentencias compuestas son ideales si no necesita parámetros ni una gran parte de lógica de procedimiento, sino sólo una mínima lógica de procedimiento, ya que la lógica necesaria está pensada principalmente para el flujo de datos. Para obtener más información acerca de las sentencias compuestas, consulte:

- "SQL Procedural Language (SQL PL) en DB2" en la página 67
- Sentencia de SQL dinámica compuesta

¿Resulta más apropiada una función de SQL que un procedimiento de SQL?
Si puede reescribir el SQL que desea incluir en el procedimiento como una

expresión en lugar de varias sentencias de SQL, probablemente es mejor que implemente la lógica como una función, porque las expresiones de SQL son más eficaces y se ejecutarán mejor que el SQL PL en combinación con sentencias de SQL. Por ejemplo, una expresión CASE se ejecutará mejor que una sentencia IF ELSE o CASE que contenga otras sentencias de SQL. Si desea un ejemplo de procedimiento de SQL reescrito eficazmente como una función de SQL, consulte:

- "Mejora del rendimiento de los procedimientos de SQL" en la página 82

* ¿Se debe utilizar este procedimiento para actividades de OLTP?

* Si el procedimiento de SQL que desea escribir se debe utilizar en una aplicación OLTP, sería una buena idea leer acerca de otras características de DB2 que puedan ayudarle a maximizar el rendimiento de la aplicación. Otras características de DB2 para descubrir son:
  - Tablas temporales globales: útiles para el almacenamiento de resultados intermedios y más rápidas en el acceso que las tablas base
  - Columnas generadas en las tablas: utilizadas para generar automáticamente valores de columna en una fila
  - Mandato RUNSTATS: utilizado para reunir estadísticas sobre tablas a fin de mejorar el rendimiento de las consultas
  - "Mejora del rendimiento de los procedimientos de SQL" en la página 82

Consideraciones de rendimiento para los procedimientos de SQL PL

El rendimiento casi siempre es importante. Si piensa implementar un procedimiento de SQL para realizar lógica compleja, como, por ejemplo, un algoritmo matemático complejo, que requiere gran parte de SQL PL y muy pocas consultas o modificaciones de base de datos, debe tomar en consideración implementar un procedimiento externo en su lugar. Si decide implementar un procedimiento de SQL PL, para obtener consejos sobre cómo escribir procedimientos de SQL PL que funcionen bien, consulte:

- "Mejora del rendimiento de los procedimientos de SQL" en la página 82

Conceptos relacionados:

- "Rutinas en el desarrollo de aplicaciones" en la página 3
- "Tipos de rutinas (procedimientos, funciones, métodos)" en la página 5
- "Niveles de acceso de SQL en rutinas de SQL" en la página 69
- "SQL Procedural Language (SQL PL) en DB2" en la página 67
- "Bloques compuestos y ámbito de las variables en los procedimientos de SQL" en la página 76

Creación de procedimientos de SQL desde la línea de mandatos

Requisitos previos:

- El usuario debe tener los privilegios necesarios para ejecutar la sentencia CREATE PROCEDURE para un procedimiento de SQL.
- Privilegios para ejecutar todas las sentencias de SQL incluidas en el cuerpo de procedimiento de SQL del procedimiento.
- Cualquier objeto de base de datos al que se haga referencia en la sentencia CREATE PROCEDURE del procedimiento de SQL debe existir antes de la ejecución de la sentencia.

Procedimiento:

- Seleccione un carácter de terminación alternativo para el Procesador de línea de mandatos (DB2 CLP), distinto del carácter de terminación por omisión, que es un punto y coma (';'), para utilizarlo en el script que preparará en el próximo paso.

  Esto es necesario para que el CLP pueda distinguir el final de las sentencias de SQL que aparecen en el cuerpo de la sentencia CREATE de una rutina, con respecto al final de la sentencia CREATE PROCEDURE propiamente dicha. El carácter de punto y coma se debe utilizar para terminar las sentencias de SQL dentro del cuerpo de la rutina de SQL, y el carácter de terminación alternativo elegido se debe utilizar para terminar la sentencia CREATE y cualquier otra sentencia de SQL que se pueda incluir dentro del script de CLP.

  Por ejemplo, en la sentencia CREATE PROCEDURE siguiente, se utiliza el signo 'at;' ('@') como carácter de terminación para un script de DB2 CLP denominado myCLPscript.db2:

```
    CREATE PROCEDURE UPDATE_SALARY_IF
    (IN employee_number CHAR(6), IN rating SMALLINT)
    LANGUAGE SQL
    BEGIN
      DECLARE not_found CONDITION FOR SQLSTATE '02000';
      DECLARE EXIT HANDLER FOR not_found
         SIGNAL SQLSTATE '20000' SET MESSAGE_TEXT = 'Employee not found';

      IF (rating = 1)
        THEN UPDATE employee
          SET salary = salary * 1.10, bonus = 1000
          WHERE empno = employee_number;
      ELSEIF (rating = 2)
        THEN UPDATE employee
          SET salary = salary * 1.05, bonus = 500
          WHERE empno = employee_number;
      ELSE UPDATE employee
          SET salary = salary * 1.03, bonus = 0
          WHERE empno = employee_number;
      END IF;
    END
@
```

- Ejecute el script de DB2 CLP que contiene la sentencia CREATE PROCEDURE para el procedimiento desde la línea de mandatos utilizando el siguiente mandato de CLP:

```
db2 -td <carácter-terminación> -vf <nombre-script-CLP>
```

  donde *<carácter-terminación>* es el carácter de terminación utilizado en el archivo de script de CLP *nombre-script-CLP* que se ha de ejecutar.

  La opción de DB2 CLP -td indica que el terminador de CLP por omisión se debe restablecer con el *carácter de terminación*. La opción *-vf* indica que se debe utilizar el elemento verboso opcional de CLP (*-v*), el cual permitirá que cada sentencia de SQL o mandato del script se visualice en la pantalla a medida que se ejecuta, junto con toda la salida que resulte de su ejecución. La opción *-f* indica que el destino del mandato es un archivo.

  Para ejecutar el script específico que se muestra en el primer paso, emita el mandato siguiente desde el indicador de mandatos del sistema:

```
db2 -td@ -vf myCLPscript.db2
```

Conceptos relacionados:

- "Rutinas en el desarrollo de aplicaciones" en la página 3

Información relacionada:

- "Sentencia CREATE TYPE (Estructurado)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE FUNCTION (Escalar de SQL, tabla o fila)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE PROCEDURE (SQL)" en la publicación *Consulta de SQL, Volumen 2*

Parámetros de los procedimientos de SQL

DB2 da soporte al uso de parámetros de entrada, parámetros de salida y parámetros de entrada y salida en los procedimientos de SQL. Las palabras clave IN, OUT e INOUT en la sentencia CREATE PROCEDURE indican la modalidad o el uso deseado del parámetro. Los parámetros IN y OUT se pasan por valor, y los parámetros INOUT se pasan por referencia.

Los parámetros son opcionales y es posible crear procedimientos de SQL que no tengan ninguno; sin embargo, cuando se especifiquen varios parámetros, deben ser exclusivos dentro del procedimiento. Además, si hay que declarar una variable en el procedimiento con el mismo nombre que un parámetro, se debe declarar en un bloque atómico etiquetado anidado dentro del procedimiento; de lo contrario, DB2 detectará que es una referencia ambigua.

Los parámetros de los procedimientos de SQL no pueden recibir el nombre de SQLSTATE ni SQLCODE, independientemente del tipo de datos del parámetro. Consulte la sentencia CREATE PROCEDURE para obtener detalles completos sobre las modalidades de los parámetros y las restricciones de éstos en los procedimientos de SQL.

El siguiente procedimiento de SQL denominado *myparams* ilustra el uso de las modalidades de los parámetros IN, INOUT y OUT. Se puede decir que el procedimiento de SQL se define en un archivo CLP denominado *myfile.db2* y que se utiliza la línea de mandatos.

```
CREATE PROCEDURE myparams (IN p1 INT, INOUT p2 INT, OUT p3 INT)
LANGUAGE SQL
BEGIN
 SET p2 = p2 + 1;
 SET p3 = 2 * p1;
END@
```

Para crear el procedimiento almacenado, entre lo siguiente desde la línea de mandatos:

```
db2 -td@ -vf myfile.db2
```

Luego, para llamar al procedimiento, entre lo siguiente desde la línea de mandatos:

```
db2 "CALL myParms(1, 3, ?)"
```

El signo '?' es un marcador de parámetro para el parámetro de salida. Deberá especificar un valor de entrada para cualquier parámetro INOUT, aunque no se haga referencia al mismo dentro del procedimiento. Se devolverá la salida siguiente:

```
Value of output parameters
--------------------------
Parameter Name  : P2
Parameter Value : 4
```

```
Parameter Name  : P3
Parameter Value : 2

Return Status = 0
```

Conceptos relacionados:

- "Estilos de parámetros para rutinas externas" en la página 95
- "Modalidades de parámetros de procedimiento" en la página 46
- "SQL Procedural Language (SQL PL) en DB2" en la página 67
- "Sentencias CREATE para las rutinas de SQL" en la página 68
- "Variables en los procedimientos de SQL (sentencias DECLARE, DEFAULT, SET)" en la página 75

Tareas relacionadas:

- "Creación de procedimientos de SQL desde la línea de mandatos" en la página 72

Variables en los procedimientos de SQL (sentencias DECLARE, DEFAULT, SET)

Las variables están soportadas en los procedimientos como parte de una sentencia compuesta. Se utiliza la palabra clave DECLARE al declarar las variables. Las declaraciones de variables deben aparecer al principio del cuerpo del procedimiento de SQL, antes de la declaración de cualquier condición, manejador de condiciones, cursor o sentencia de SQL.

Las variables se pueden declarar con un valor por omisión utilizando la cláusula DEFAULT, donde el valor por omisión puede ser una constante, un valor de registro especial o una expresión. Por ejemplo:

```
CREATE PROCEDURE P2(INOUT a VARCHAR(8),
                    OUT  b INTEGER)
LANGUAGE SQL
BEGIN
  DECLARE var1 INTEGER DEFAULT 0;
  DECLARE var2 VARCHAR(5) DEFAULT  a || 'bc';

   -- otras sentencias de SQL --

END@
```

Bajo las declaraciones de variables de un procedimiento de SQL, es posible asignar valores a las variables y los parámetros, incluidos los parámetros de entrada, utilizando la sentencia de asignación del modo siguiente:

```
CREATE PROCEDURE P2(INOUT a VARCHAR(8),
                    OUT  b INTEGER)
LANGUAGE SQL
BEGIN
  DECLARE var1 INTEGER DEFAULT 0;
  DECLARE var2 VARCHAR(5) DEFAULT  a || 'bc';

   SET var1 = 0;
   SET var1 = var1 + 1;
   SET var2  = var2 || 'def';

   SET a = var1;
   SET b = var2;
END@
```

Conceptos relacionados:

- "SQL Procedural Language (SQL PL) en DB2" en la página 67
- "Sentencias CREATE para las rutinas de SQL" en la página 68
- "Parámetros de los procedimientos de SQL" en la página 74
- "Mejora del rendimiento de los procedimientos de SQL" en la página 82

Tareas relacionadas:

- "Creación de procedimientos de SQL desde la línea de mandatos" en la página 72

Bloques compuestos y ámbito de las variables en los procedimientos de SQL

Dentro de un procedimiento de SQL, puede tener una o más sentencias compuestas. Las sentencias compuestas introducen un bloque de sentencias de SQL que se compilan y ejecutan como una sola sentencia en DB2. Las sentencias compuestas se reconocen fácilmente porque empiezan y finalizan con las palabras clave BEGIN y END y se pueden etiquetar para identificar el bloque de código.

El uso de una etiqueta se vuelve importante en el contexto del ámbito de las variables, ya que se puede utilizar para calificar los nombres de las variables, lo cual importa en la identificación y referencia de las variables en las distintas sentencias compuestas o en sentencias compuestas anidadas.

En el ejemplo siguiente, hay dos declaraciones de la variable *a*. Una instancia de la misma se declara en la sentencia compuesta externa que está etiquetada como *lab1*, y la segunda instancia se declara en la sentencia compuesta interna etiquetada como *lab2*. Tal como se ha escrito, DB2 supondrá que la referencia a la variable *a* en la sentencia de asignación es la que está en el ámbito local del bloque compuesto, con la etiqueta *lab2*. No obstante, si la instancia pretendida de la variable *a* era la declarada en el bloque de sentencias compuestas con la etiqueta "lab1", para que su referencia sea correcta en el bloque compuesto más interno, la variable se debería haber calificado con la etiqueta de ese bloque. Es decir, calificada como: *lab1.a*.

```
CREATE PROCEDURE P1 ()
LANGUAGE SQL
  lab1: BEGIN
    DECLARE a INT DEFAULT 100;
    lab2: BEGIN
      DECLARE a INT DEFAULT NULL;

      SET a = a + lab1.a;

      UPDATE T1
       SET T1.b = 5
        WHERE T1.b = a;  -- La variable a hace referencia a lab2.a
                         -- a menos que se califique de otra manera
    lab2: END;
 END lab1@
```

La sentencia compuesta más remota de un procedimiento de SQL se puede declarar como atómica añadiendo la palabra clave ATOMIC después de la palabra clave BEGIN. Si se produce algún error en la ejecución de las sentencias que abarca la sentencia compuesta atómica, se retrotrae toda la sentencia compuesta.

Conceptos relacionados:

- "SQL Procedural Language (SQL PL) en DB2" en la página 67
- "Parámetros de los procedimientos de SQL" en la página 74
- "Variables en los procedimientos de SQL (sentencias DECLARE, DEFAULT, SET)" en la página 75

Tareas relacionadas:

- "Creación de procedimientos de SQL desde la línea de mandatos" en la página 72

Devolución de mensajes de error desde procedimientos de SQL

Cuando se emite una sentencia CREATE PROCEDURE para un procedimiento de SQL, DB2 puede aceptar la sintaxis del cuerpo del procedimiento de SQL, pero puede fallar en la creación del procedimiento de SQL durante las etapas de precompilación o compilación. En estas situaciones, DB2 crea normalmente un archivo de registro que contiene los mensajes de error.

Para recuperar los mensajes de error generados por DB2 y el compilador C para un procedimiento de SQL, visualice el archivo de registro de mensajes del directorio siguiente del servidor de bases de datos:

UNIX *instancia*/function/routine/sqlproc/*nombre_bd*/*nombre_esquema*/tmp

donde *instancia* representa la vía de acceso de la instancia de DB2, *nombre_bd* representa el alias de base de datos, y *nombre_esquema* representa el esquema con el que se ha emitido la sentencia CREATE PROCEDURE.

Windows
instancia\function\routine\sqlproc*nombre_bd**nombre_esquema*\tmp

donde *instancia* representa la vía de acceso de la instancia de DB2, *nombre_bd* representa el alias de base de datos, y *nombre_esquema* representa el esquema con el que se ha emitido la sentencia CREATE PROCEDURE.

Nota: Si el nombre de esquema del procedimiento de SQL no se emite como parte de la sentencia CREATE PROCEDURE, DB2 utiliza el valor del registro especial CURRENT SCHEMA. Para visualizar el valor del registro especial CURRENT SCHEMA, emita la sentencia siguiente en el CLP:

```
VALUES CURRENT SCHEMA
```

Tareas relacionadas:

- "Depuración de rutinas" en la página 42

Información relacionada:

- "Registro especial CURRENT SCHEMA" en la publicación *Consulta de SQL, Volumen 1*

Manejadores de condiciones en los procedimientos de SQL

Los apartados siguientes describen los manejadores de condiciones y cómo se pueden utilizar para permitir que los procedimientos de SQL reaccionen ante diversas condiciones de una base de datos.

Manejadores de condiciones en los procedimientos de SQL

Los manejadores de condiciones determinan el comportamiento del procedimiento de SQL cuando se produce una condición. Puede declarar uno o más manejadores de condiciones en el procedimiento de SQL para condiciones generales, condiciones nombradas o valores específicos de SQLSTATE.

Si una sentencia del procedimiento de SQL produce una condición SQLWARNING o NOT FOUND y se ha declarado un manejador para la condición respectiva, DB2® pasa el control al manejador correspondiente. Si no se ha declarado un manejador para una condición de ese tipo, DB2 pasará el control a la siguiente sentencia del cuerpo del procedimiento de SQL. Si se han declarado las variables SQLCODE y SQLSTATE, éstas contendrán los valores correspondientes a la condición.

Si una sentencia del procedimiento de SQL produce una condición SQLEXCEPTION, y si se ha declarado un manejador para la condición SQLEXCEPTION o el SQLSTATE específico, DB2 pasa el control a dicho manejador. Si se han declarado las variables SQLSTATE y SQLCODE, sus valores después de la ejecución satisfactoria de un manejador serán, respectivamente, '00000' y 0.

Si una sentencia del procedimiento de SQL produce una condición SQLEXCEPTION y no se ha declarado un manejador para la condición SQLEXCEPTION o el SQLSTATE específico, DB2 interrumpe el procedimiento de SQL y vuelve al llamante.

Conceptos relacionados:

- "Sentencias SIGNAL y RESIGNAL en manejadores de condiciones" en la página 81
- "Declaraciones de manejadores de condiciones" en la página 78
- "Variables SQLCODE y SQLSTATE en procedimientos de SQL" en la página 81

Tareas relacionadas:

- "Devolución de mensajes de error desde procedimientos de SQL" en la página 77

Declaraciones de manejadores de condiciones

Para poder definir el comportamiento del procedimiento de SQL cuando se produzcan determinadas condiciones, tiene que declarar manejadores de condiciones. El formato general de la declaración de un manejador es:

```
  DECLARE tipo-manejador  HANDLER FOR  condición
sentencia-procedimiento-SQL
```

Cuando DB2® produce una condición que coincide con *condición*, DB2 pasa el control al manejador de condiciones. El manejador de condiciones realiza la acción indicada por el *tipo-manejador* y, a continuación, ejecuta la *sentencia-procedimiento-SQL*.

Tipos-manejadores

CONTINUE
Especifica que, una vez que finalice la *sentencia-procedimiento-SQL*, la ejecución continuará con la sentencia que sigue a la que ha ocasionado el error.

EXIT Especifica que, una vez que finalice la *sentencia-procedimiento-SQL*, la ejecución continuará al final de la sentencia compuesta que contiene el manejador.

UNDO
Especifica que, antes de que se ejecute la *sentencia-procedimiento-SQL*, DB2 retrotraerá las posibles operaciones de SQL que se hayan producido en la sentencia compuesta que contiene el manejador. Una vez que finalice la *sentencia-procedimiento-SQL*, la ejecución continuará al final de la sentencia compuesta que contiene el manejador.

Nota: Sólo se pueden declarar manejadores UNDO en las sentencias compuestas ATOMIC.

Condiciones
DB2 proporciona tres condiciones generales:

NOT FOUND
Identifica cualquier condición que tenga como consecuencia un SQLCODE de +100 o un SQLSTATE que empiece por los caracteres '02'.

SQLEXCEPTION
Identifica cualquier condición que tenga como consecuencia un SQLCODE negativo.

SQLWARNING
Identifica cualquier condición que tenga como consecuencia una condición de aviso (SQLWARN0 sea 'W'), o un código de retorno de SQL positivo distinto de +100. El valor del SQLSTATE correspondiente empezará por los caracteres '01'.

También puede utilizar la sentencia DECLARE para definir su propia condición para un SQLSTATE específico.

sentencia-procedimiento-SQL
Puede utilizar una sola sentencia de procedimiento de SQL para definir el comportamiento del manejador de condiciones. DB2 acepta una sentencia compuesta delimitada por un bloque BEGIN...END como una sola sentencia de procedimiento de SQL. Si utiliza una sentencia compuesta para definir el comportamiento de un manejador de condiciones, y si desea que el manejador conserve el valor de las variables SQLSTATE o SQLCODE, debe asignar el valor de la variable a un parámetro o una variable local en la primera sentencia del bloque compuesto. Si la primera sentencia de un bloque compuesto no asigna el valor de SQLSTATE o SQLCODE a un parámetro o una variable local, SQLSTATE y SQLCODE no pueden conservar el valor que ha ocasionado que DB2 invocara al manejador de condiciones.

Los ejemplos siguientes muestran manejadores de condiciones simples:

Manejador CONTINUE
Este manejador asigna el valor de 1 a la variable local *at_end* cuando DB2 produce una condición NOT FOUND. Luego, DB2 pasa el control a la sentencia que sigue a la que ha producido la condición NOT FOUND.

```
DECLARE CONTINUE HANDLER FOR NOT FOUND SET at_end = 1;
```

Manejador EXIT
En este ejemplo, el ámbito del manejador de salida se reduce a la sentencia compuesta etiquetada como A. Si la tabla JAVELIN no existe, la sentencia DROP producirá la condición NO_TABLE. Se activará el manejador de

salida, OUT_BUFFER se establecerá en la serie 'La tabla no existe' y la ejecución continuará con la sentencia INSERT de C, sin visitar ninguna otra sentencia de la sentencia compuesta A. Si la sentencia DROP finaliza satisfactoriamente, no se activará el manejador y la ejecución continuará con la sentencia SET de B.

```
CREATE PROCEDURE EXIT_TEST ()
 LANGUAGE SQL
          BEGIN
    DECLARE OUT_BUFFER VARCHAR(80);
    DECLARE NO_TABLE CONDITION FOR SQLSTATE '42704';

   A: BEGIN
        DECLARE EXIT HANDLER FOR NO_TABLE
          BEGIN
             SET OUT_BUFFER='La tabla no existe';
    END;

          -- Eliminar la tabla que potencialmente no existe:
          DROP TABLE JAVELIN;

       B: SET OUT_BUFFER='La tabla se ha eliminado satisfactoriamente';
    END;

    -- Copiar OUT_BUFFER a alguna tabla de mensajes:
    C: INSERT INTO MESSAGES VALUES OUT_BUFFER;
 END
```

Manejador UNDO

En este ejemplo, el ámbito del manejador para deshacer se reduce a la sentencia compuesta etiquetada como A. Si la tabla JAVELIN no existe, la sentencia DROP producirá la condición NO_TABLE. Se activará el manejador para deshacer, se retrotraerá la sentencia INSERT que precede a DROP, OUT_BUFFER se establecerá en la serie 'La tabla no existe' y la ejecución continuará con la sentencia INSERT de C, sin visitar ninguna otra sentencia de la sentencia compuesta A. Si la sentencia DROP finaliza satisfactoriamente, no se activará el manejador y la ejecución continuará con la sentencia SET de B.

```
CREATE PROCEDURE UNDO_TEST ()
 LANGUAGE SQL
          BEGIN
    DECLARE OUT_BUFFER VARCHAR(80);
    DECLARE NO_TABLE CONDITION FOR SQLSTATE '42704';

    A: BEGIN ATOMIC
          DECLARE UNDO HANDLER FOR NO_TABLE
          BEGIN
             SET OUT_BUFFER='La tabla no existe';
    END;

          INSERT INTO MESSAGES VALUES
             'Este mensaje será eliminado por una retrotracción.';

          -- Eliminar la tabla que potencialmente no existe:
          DROP TABLE JAVELIN;

       B: SET OUT_BUFFER='La tabla se ha eliminado satisfactoriamente';
    END;

    -- Copiar OUT_BUFFER a alguna tabla de mensajes:
    C: INSERT INTO MESSAGES VALUES OUT_BUFFER;
 END
```

Nota: Sólo se pueden declarar manejadores UNDO en las sentencias compuestas ATOMIC.

Conceptos relacionados:

- "Manejadores de condiciones en los procedimientos de SQL" en la página 78
- "Sentencias SIGNAL y RESIGNAL en manejadores de condiciones" en la página 81
- "Variables SQLCODE y SQLSTATE en procedimientos de SQL" en la página 81

Información relacionada:

- "Sentencia SQL compuesto (Incorporado)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE PROCEDURE" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia SQL compuesto (Dinámico)" en la publicación *Consulta de SQL, Volumen 2*

Sentencias SIGNAL y RESIGNAL en manejadores de condiciones

Puede utilizar las sentencias SIGNAL y RESIGNAL para producir explícitamente un SQLSTATE específico. Utilice la cláusula SET MESSAGE_TEXT de las sentencias SIGNAL y RESIGNAL para definir el texto que DB2® visualizará junto con el SQLSTATE producido.

En el ejemplo siguiente, el cuerpo del procedimiento de SQL declara un manejador de condiciones para el SQLSTATE personalizado 72822. Cuando el procedimiento de SQL ejecuta la sentencia SIGNAL que produce el SQLSTATE 72822, DB2 invoca al manejador de condiciones. El manejador de condiciones comprueba el valor de la variable *var* de SQL mediante una sentencia IF. Si *var* es OK, el manejador redefine el valor de SQLSTATE como 72623 y asigna un literal de serie al texto asociado al SQLSTATE 72623. Si *var* no es OK, el manejador redefine el valor de SQLSTATE como 72319 y asigna el valor de *var* al texto asociado a ese SQLSTATE.

```
DECLARE EXIT HANDLER FOR SQLSTATE '72822'
    BEGIN
        IF ( var = 'OK' )
    RESIGNAL SQLSTATE '72623' SET MESSAGE_TEXT = 'Got SQLSTATE 72822';
        ELSE
    RESIGNAL SQLSTATE '72319' SET MESSAGE_TEXT = var;
    END;

 SIGNAL SQLSTATE '72822';
```

Conceptos relacionados:

- "Manejadores de condiciones en los procedimientos de SQL" en la página 78
- "Declaraciones de manejadores de condiciones" en la página 78
- "Variables SQLCODE y SQLSTATE en procedimientos de SQL" en la página 81

Información relacionada:

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

Variables SQLCODE y SQLSTATE en procedimientos de SQL

Como ayuda para depurar los procedimientos de SQL, le puede resultar útil insertar el valor de SQLCODE y SQLSTATE en una tabla en diversos puntos del procedimiento de SQL, o devolver los valores de SQLCODE y SQLSTATE en una

serie de diagnóstico como parámetro de salida (OUT). Para utilizar los valores de SQLCODE y SQLSTATE, debe declarar las variables de SQL siguientes en el cuerpo del procedimiento de SQL:

```
DECLARE SQLCODE INTEGER DEFAULT 0;
DECLARE SQLSTATE CHAR(5) DEFAULT '00000';
```

DB2® establece implícitamente estas variables siempre que se ejecuta una sentencia. Si una sentencia produce una condición para la que existe un manejador, los valores de las variables SQLSTATE y SQLCODE estarán disponibles al iniciarse la ejecución del manejador. No obstante, las variables se restablecerán tan pronto como se ejecute la primera sentencia del manejador. Por lo tanto, es una práctica habitual copiar los valores de SQLSTATE y SQLCODE en variables locales en la primera sentencia del manejador. En el ejemplo siguiente, se utiliza un manejador CONTINUE para cualquier condición a fin de copiar la variable SQLCODE en otra variable denominada `retcode`. La variable `retcode` se puede utilizar entonces en las sentencias ejecutables para controlar la lógica de procedimiento o devolver el valor como parámetro de salida.

```
BEGIN
  DECLARE SQLCODE INTEGER DEFAULT 0;
  DECLARE retcode INTEGER DEFAULT 0;

  DECLARE CONTINUE HANDLER FOR SQLEXCEPTION, SQLWARNING, NOT FOUND
     SET retcode = SQLCODE;

  sentencias-ejecutables
END
```

Nota: Cuando se accede a las variables SQLCODE o SQLSTATE en un procedimiento de SQL, DB2 establece el valor de SQLCODE en 0 y el de SQLSTATE en '00000' para la sentencia posterior.

Conceptos relacionados:

- "Manejadores de condiciones en los procedimientos de SQL" en la página 78
- "Sentencias SIGNAL y RESIGNAL en manejadores de condiciones" en la página 81
- "Declaraciones de manejadores de condiciones" en la página 78

Mejora del rendimiento de los procedimientos de SQL

Visión general de cómo DB2 compila SQL PL y SQL PL en línea:

Antes de explicar cómo se mejora el rendimiento de los procedimientos de SQL, habría que explicar cómo los compila DB2 después de la ejecución de la sentencia CREATE PROCEDURE.

Cuando se crea un procedimiento de SQL, DB2 separa las consultas de SQL del cuerpo del procedimiento con respecto a la lógica del procedimiento. Para maximizar el rendimiento, las consultas de SQL se compilan estáticamente en secciones de un paquete. En el caso de una consulta compilada estáticamente, una sección consta principalmente del plan de acceso seleccionado por el optimizador de DB2 para esa consulta. Un paquete es una colección de secciones. Si desea más información sobre los paquetes y secciones, vea la Consulta de SQL para DB2. La lógica del procedimiento se compila en una biblioteca enlazada de forma dinámica.

Durante la ejecución de un procedimiento, cada vez que fluye el control desde la lógica del mismo a una sentencia de SQL, existe una "conmutación de contexto"

entre la DLL y el mecanismo de DB2. A partir de DB2 Versión 8.1, los procedimientos de SQL se ejecutan en la ″modalidad no protegida″. Es decir, se ejecutan en el mismo espacio de direcciones que el mecanismo de DB2. Por lo tanto, la conmutación de contexto a la que se hace referencia aquí no es una conmutación de contexto total al nivel del sistema operativo, sino un cambio de capa dentro de DB2. La reducción del número de conmutaciones de contexto en los procedimientos que se invocan muy a menudo, como los procedimientos en una aplicación OLTP, o que procesan un número elevado de filas, como los procedimientos que realizan el borrado de datos, puede tener un efecto notable en su rendimiento.

Mientras que un procedimiento de SQL que contiene SQL PL se implementa compilando estáticamente sus consultas de SQL individuales en secciones de un paquete, una función de SQL PL en línea se implementa, como su nombre sugiere, colocando en línea el cuerpo de la función en la consulta que la utiliza. Las consultas de las funciones de SQL se compilan conjuntamente, como si el cuerpo de la función fuese una sola consulta. La compilación se produce cada vez que se compila una sentencia que utiliza la función. A diferencia de lo que sucede en los procedimientos de SQL, las sentencias de procedimiento de las funciones de SQL no se ejecutan en una capa distinta de las sentencias de flujo de datos. Por lo tanto, no se produce conmutación de contexto cada vez que fluye el control desde una sentencia de procedimiento a una sentencia de flujo de datos o viceversa.

Si no existen efectos secundarios en la lógica, utilice una función de SQL en su lugar:

A causa de la diferencia de compilación entre el SQL PL en los procedimientos y el SQL PL en línea en las funciones, es razonable suponer que una parte del código de procedimiento se ejecutará más rápido en una función que en un procedimiento si sólo consulta datos de SQL y no efectúa modificaciones de datos - es decir, no tiene efectos secundarios en los datos incluidos en la base de datos o externos a ésta.

Esto sólo es positivo si todas las sentencias que tiene que ejecutar están soportadas en las funciones de SQL. Las funciones de SQL no pueden contener sentencias de SQL que modifiquen la base de datos. Además, sólo un subconjunto del SQL PL está disponible en el SQL PL en línea de las funciones. Por ejemplo: no puede ejecutar sentencias CALL, declarar cursores ni devolver conjuntos de resultados en las funciones de SQL.

A continuación, se proporciona un ejemplo de un procedimiento de SQL que contiene SQL PL y que era un buen candidato para la conversión a una función de SQL a fin de maximizar el rendimiento:

```
CREATE PROCEDURE GetPrice (IN Vendor CHAR&(20&),
                           IN Pid INT, OUT price DECIMAL(10,3))
LANGUAGE SQL
BEGIN
  IF Vendor eq; ssq;Vendor 1ssq;
    THEN SET price eq; (SELECT ProdPrice
                         FROM V1Table
                         WHERE Id = Pid);
  ELSE IF Vendor eq; ssq;Vendor 2ssq;
    THEN SET price eq; (SELECT Price FROM V2Table
                         WHERE Pid eq; GetPrice.Pid);
  END IF;
END
```

Aquí está la función de SQL reescrita:

```
CREATE FUNCTION GetPrice (Vendor CHAR(20), PId INT)
RETURNS  DECIMAL(10,3)
LANGUAGE SQL
BEGIN
  DECLARE price DECIMAL(10,3);
  IF Vendor = 'Vendor 1'
    THEN SET price = (SELECT ProdPrice
                             FROM V1Table
                             WHERE Id = Pid);
  ELSE IF Vendor = 'Vendor 2'
    THEN SET price = (SELECT Price FROM V2Table
                WHERE Pid = GetPrice.Pid);
  END IF;
  RETURN price;
END
```

Recuerde que la invocación de una función es distinta de la de un procedimiento. Para invocar la función, utilice la sentencia VALUES o invoque la función donde sea válida una expresión, como en una sentencia SELECT o SET. Cualquiera de las siguientes son formas válidas de invocar la nueva función:

```
VALUES (GetPrice('IBM', 324))

SELECT VName FROM Vendors WHERE GetPrice(Vname, Pid) < 10

SET price = GetPrice(Vname, Pid)
```

Evite varias sentencias en un procedimiento de SQL PL cuando sólo una sea suficiente:

Aunque generalmente resulta una buena idea escribir SQL conciso, es muy fácil olvidarse de ello en la práctica. Por ejemplo, las sentencias de SQL siguientes:

```
INSERT INTO tab_comp VALUES (item1, price1, qty1);
INSERT INTO tab_comp VALUES (item2, price2, qty2);
INSERT INTO tab_comp VALUES (item3, price3, qty3);
```

se pueden reescribir como una sola sentencia:

```
INSERT INTO tab_comp VALUES (item1, price1, qty1),
                            (item2, price2, qty2),
                            (item3, price3, qty3);
```

La inserción múltiple de filas requerirá aproximadamente una tercera parte del tiempo necesario para ejecutar las tres sentencias originales. De forma aislada, esta mejora puede parecer poco importante, pero si el fragmento de código se ejecuta repetidamente, por ejemplo, en un bucle o en un cuerpo de activador, la mejora puede ser significativa.

De modo similar, una secuencia de sentencias SET como la siguiente:

```
SET A = expr1;
SET B = expr2;
SET C = expr3;
```

se puede escribir como una sola sentencia VALUES:

```
VALUES expr1, expr2, expr3 INTO A, B, C;
```

Esta transformación conserva la semántica de la secuencia original si no existen dependencias entre dos sentencias. Para ilustrar esto, tome en consideración lo siguiente:

```
SET A = monthly_avg * 12;
SET B = (A / 2) * correction_factor;
```

La conversión de las dos sentencias anteriores a:

```
VALUES (monthly_avg * 12, (A / 2) * correction_factor) INTO A, B;
```

no conserva la semántica original porque las expresiones anteriores a la palabra clave INTO se evalúan 'en paralelo'. Esto significa que el valor asignado a *B* no está basado en el valor asignado a *A*, lo que era la semántica deseada de las sentencias originales.

Reduzca varias sentencias de SQL a una sola expresión de SQL:

Al igual que otros lenguajes de programación, el lenguaje SQL proporciona dos tipos de construcciones condicionales: de procedimiento (sentencias IF y CASE) y funcionales (expresiones CASE). En la mayoría de las circunstancias en las que se puede utilizar cualquier tipo para expresar un cálculo, utilizar uno u otro es cuestión de preferencias. Sin embargo, la lógica escrita empleando expresiones CASE no sólo es más compacta, sino también más eficaz que la lógica escrita empleando sentencias CASE o IF.

Tome en consideración el siguiente fragmento de código SQL PL:

```
IF (Price <= MaxPrice) THEN
  INSERT INTO tab_comp(Id, Val) VALUES(Oid, Price)semi;
ELSE
  INSERT INTO tab_comp(Id, Val) VALUES(Oid, MaxPrice)semi;
END IF;
```

La condición de la cláusula IF sólo se utiliza para decidir qué valor se inserta en la columna tab_comp.Val. Para evitar la conmutación de contexto entre la capa de procedimiento y la capa de flujo de datos, se puede expresar la misma lógica como una sola inserción (INSERT) con una expresión CASE:

```
INSERT INTO tab_comp(Id, Val)
       VALUES(Oid,
    CASE
               WHEN (Price <= MaxPrice) THEN Price
               ELSE MaxPrice
           END);
```

Merece la pena tener en cuenta que las expresiones CASE se pueden utilizar en cualquier contexto donde se espere un valor escalar. En particular, se pueden utilizar a la derecha de las asignaciones. Por ejemplo:

```
IF (Name IS NOT NULL) THEN
  SET ProdName = Name;
ELSEIF (NameStr IS NOT NULL) THEN
  SET ProdName = NameStr;
ELSE
  SET ProdName = DefaultName;
END IF;
```

se puede reescribir como:

```
SET ProdName = (CASE
                  WHEN (Name IS NOT NULL) THEN Name
                  WHEN (NameStr IS NOT NULL) THEN NameStr
                  ELSE  DefaultName
                END);
```

De hecho, este ejemplo determinado admite una solución incluso mejor:

```
SET ProdName = COALESCE(Name, NameStr, DefaultName);
```

No subestime el beneficio que supone tomar un tiempo para analizar el SQL y reescribirlo si es necesario. Los beneficios en el rendimiento le compensarán muchas veces el tiempo invertido en analizar y reescribir el procedimiento.

Aproveche la semántica de SQL establecida a la vez:

Las construcciones de procedimiento tales como bucles, asignaciones y cursores permiten expresar cálculos que no sería posible expresar utilizando únicamente sentencias de SQL DML. Pero, cuando se tienen sentencias de procedimiento disponibles, existe el riesgo de acudir a ellas aunque el cálculo a mano se pueda expresar, de hecho, utilizando sólo sentencias de SQL DML. Como se ha mencionado antes, el rendimiento de un cálculo de procedimiento puede consistir en órdenes de magnitud más lenta que el rendimiento de un cálculo equivalente expresado mediante sentencias de DML. Tome en consideración el siguiente fragmento de código:

```
DECLARE cur1 CURSOR FOR SELECT col1, col2 FROM tab_comp;
OPEN cur1;
FETCH cur1 INTO v1, v2;
WHILE SQLCODE ≠ 100 DO
  IF (v1 > 20) THEN
    INSERT INTO tab_sel VALUES (20, v2);
ELSE
    INSERT INTO tab_sel VALUES (v1, v2);
  END IF;
  FETCH cur1 INTO v1, v2;
END WHILE;
```

Para empezar, el cuerpo del bucle se puede mejorar aplicando la transformación explicada en el último apartado - "Reduzca varias sentencias de SQL a una sola expresión de SQL":

```
DECLARE cur1 CURSOR FOR SELECT col1, col2 FROM tab_comp;
OPEN cur1;
FETCH cur1 INTO v1, v2;
WHILE SQLCODE ≠ 100 DO
  INSERT INTO tab_sel VALUES (CASE
                                WHEN v1 > 20 THEN 20
                                ELSE v1
                              END, v2);
  FETCH cur1 INTO v1, v2;
END WHILE;
```

Pero, después de una inspección más minuciosa, es posible escribir el bloque completo de código como una inserción (INSERT) con una subselección (sub-SELECT):

```
INSERT INTO tab_sel (SELECT (CASE
                               WHEN col1 > 20 THEN 20
                               ELSE col1
                             END),
                             col2
                     FROM tab_comp);
```

En la formulación original, había una conmutación de contexto entre la capa de procedimiento y la capa de flujo de datos para cada fila de las sentencias SELECT. En la última formulación, no existe ninguna conmutación de contexto, y el optimizador tiene una oportunidad para optimizar globalmente el cálculo total.

Por otra parte, esta simplificación dramática no habría sido posible si cada sentencia INSERT estuviera destinada a una tabla distinta, como se muestra seguidamente:

```
DECLARE cur1 CURSOR FOR SELECT col1, col2 FROM tab_comp;
OPEN cur1;
FETCH cur1 INTO v1, v2;
WHILE SQLCODE ≠ 100 DO
  IF (v1 > 20) THEN
    INSERT INTO tab_default VALUES (20, v2);
ELSE
    INSERT INTO tab_sel VALUES (v1, v2);
  END IF;
  FETCH cur1 INTO v1, v2;
END WHILE;
```

Sin embargo, la naturaleza establecida a la vez de SQL también se puede aprovechar aquí:

```
INSERT INTO tab_sel (SELECT col1, col2
                     FROM tab_comp
                     WHERE col1 <= 20);
INSERT INTO tab_default (SELECT col1, col2
                         FROM tab_comp
                         WHERE col1 > 20);
```

Con miras a mejorar el rendimiento de la lógica de procedimiento existente, probablemente dará resultado emplear tiempo en eliminar bucles de cursor.

Mantenga informado al optimizador de DB2:

Cuando se crea un procedimiento, sus consultas de SQL individuales se compilan en secciones de un paquete. El optimizador de DB2 elige un plan de ejecución para una consulta basándose, entre otros elementos, en estadísticas de tabla (por ejemplo, tamaños de tabla o la frecuencia relativa de valores de datos en una columna) y en los índices disponibles en el momento en que se compila la consulta. Cuando las tablas experimentan cambios significativos, puede ser buena idea permitir que DB2 recoja estadísticas sobre estas tablas otra vez. Y, cuando se actualizan las estadísticas o cuando se crean nuevos índices, también puede ser buena idea volver a enlazar los paquetes asociados con los procedimientos de SQL que utilizan las tablas, a fin de permitir que DB2 cree planes que aprovechen las estadísticas y los índices más recientes.

Las estadísticas de tabla se pueden actualizar mediante el mandato RUNSTATS. Para volver a enlazar el paquete asociado con un procedimiento de SQL, puede utilizar el procedimiento incorporado REBIND_ROUTINE_PACKAGE que está disponible en DB2 Versión 8.1. Por ejemplo, se puede utilizar el mandato siguiente para volver a enlazar el paquete del procedimiento MYSCHEMA.MYPROC:

```
CALL SYSPROC.REBIND_ROUTINE_PACKAGE('P', 'MYSCHEMA.MYPROC', 'ANY')
```

donde 'P' indica que el paquete corresponde a un procedimiento y 'ANY' indica que cualquiera de las funciones y los tipos de la vía de acceso de SQL se toma en consideración para la resolución de la función y del tipo. Vea la entrada incluida en la Consulta de mandatos que hace referencia al mandato REBIND si desea información más detallada.

Conceptos relacionados:

- "Tipos de rutinas (procedimientos, funciones, métodos)" en la página 5
- "SQL Procedural Language (SQL PL) en DB2" en la página 67

- "Consideraciones de diseño para los procedimientos de SQL" en la página 71

Tareas relacionadas:

- "Creación de procedimientos de SQL desde la línea de mandatos" en la página 72

Funciones de tabla de SQL

Funciones de tabla de SQL que modifican datos de SQL

Cuando se especifica la cláusula MODIFIES SQL DATA en la sentencia CREATE FUNCTION de una función de tabla de SQL, el cuerpo de la función de tabla de SQL puede incluir sentencias de SQL que modifiquen los datos de tabla. Todas las sentencias soportadas en una sentencia dinámica compuesta están permitidas en el cuerpo de una función de tabla de SQL, incluidas:

- INSERT
- UPDATE
- DELETE
- MERGE
- SELECT, cuando la cláusula FROM hace referencia a una sentencia de cambio de datos

Una función de tabla de SQL que modifica datos de SQL es una forma útil de encapsular trabajo que modifica datos y devuelve un conjunto de resultados. El conjunto de resultados se puede utilizar para devolver las filas objeto de acceso o modificación en la función de tabla. Se pueden devolver filas modificadas de diversas tablas en un solo conjunto de resultados. Una función de tabla de SQL que modifica datos de SQL se puede utilizar para comprobar las transacciones que acceden a datos de tabla o los modifican.

El soporte de la cláusula MODIFIES SQL DATA está limitado a funciones y procedimientos de SQL. No se puede crear una función de tabla externa que especifique MODIFIES SQL DATA.

Una función de tabla de SQL que modifica datos de SQL:

- sólo puede tener referencia en la cláusula FROM más remota de una selección completa intercalada en una sentencia SELECT, SELECT INTO, SET o RETURN.
- debe ser la única función de tabla de SQL que modifique datos de SQL dentro de una cláusula FROM de una sentencia SELECT.
- debe aparecer como la última referencia de tabla en la cláusula FROM cuando existen diversas referencias de tabla.
- se debe correlacionar con las otras referencias de tabla de la cláusula FROM.
- no puede tener referencia en el cuerpo de una definición de vista.
- se puede anidar dentro de una rutina si la rutina modifica datos de SQL o dentro de un activador AFTER.

Al igual que sucede con todas las rutinas, una función de tabla sólo se puede invocar satisfactoriamente si el definidor de la función de tabla está autorizado para ejecutar todas las sentencias de SQL del cuerpo de la función.

Estas restricciones aseguran una evaluación determinista de las funciones de tabla y tablas en la sentencia. Las referencias de tabla que precedan a una función de

tabla de SQL se evaluarán completamente antes de que se ejecute la función de tabla de SQL. Las referencias de tabla incluidas en la lista SELECT o cláusula WHERE de la sentencia SELECT se evaluarán una vez completada la ejecución de la función de tabla de SQL.

Ejemplos de funciones de tabla de SQL que modifican datos de SQL:

Nota: Para ver el SQL de requisito previo completo asociado con estos ejemplos o para ejecutar un ejemplo de SQL relacionado, consulte el ejemplo tbfnuse.db2 y el script de requisito previo tbfn.db2.

Ejemplo 1: Función de tabla de SQL que modifica datos de SQL:

Esta función de tabla actualiza la cantidad de un elemento en una tabla de inventario. Se utiliza una sentencia UPDATE para actualizar la cantidad del elemento especificado mediante `itemNo` de la tabla `Inventory` con la cantidad especificada mediante `amount`. Se devuelve un conjunto de resultados que contiene el nombre del producto y la nueva cantidad del elemento. Observe que se utiliza la cláusula `MODIFIES SQL DATA` porque la función actualiza datos de tabla.

```
CREATE FUNCTION updateInv(itemNo VARCHAR(20), amount INTEGER)
   RETURNS TABLE (productName varchar(20),
                  quantity INTEGER)
   LANGUAGE SQL
   MODIFIES SQL DATA
   BEGIN ATOMIC

     UPDATE Inventory as I
       SET quantity = quantity + amount
         WHERE I.itemID = itemNo;

     RETURN
       SELECT I.itemName, I.quantity
         FROM Inventory as I
           WHERE I.itemID = itemNo;
   END
```

Ejemplo 2: Invocación de una función de tabla de SQL que modifica datos de SQL:

La función de tabla de SQL del Ejemplo 1 se invoca desde una sentencia SELECT. La cantidad de un elemento identificado mediante el número de elemento, 'ISBN-0-8021-3424-6', aumenta en cinco (5). Se devuelven el nombre del producto y la cantidad actualizada del elemento.

```
SELECT productName, quantity
  FROM TABLE(updateInv('ISBN-0-8021-3424-6', 5)) AS T


PRODUCTNAME          QUANTITY
-------------------- -----------
Feng Shui at Home             15
```

Ejemplo 3: Invocación de una función de tabla de SQL que modifica datos de SQL correlacionada con otra referencia de tabla:

En este ejemplo, se actualizan las cantidades de diversos elementos existentes en la tabla de inventario 'Inventory'. La cláusula VALUES se utiliza para generar la referencia de tabla, 'newItem', que contiene las filas de los elementos que se han de actualizar. La función de tabla 'updateInv' se correlaciona con la referencia de tabla 'newItem', ya que, como mínimo, una columna de 'newItem' aparece como

argumento para la función de tabla 'updateInv'. Observe que la función de tabla es la última referencia de tabla de la cláusula FROM.

```
SELECT newItem.id, TF.productName, TF.quantity
     FROM (VALUES ('ISBN-0-8021-3424-6', 2),
                  ('ISBN-0-8021-4612-1', 5)) AS newItem(id, quantity),
           TABLE(updateInv(newItem.id, newItem.quantity)) AS TF


ID                 PRODUCTNAME          QUANTITY
------------------ -------------------- -----------
ISBN-0-8021-3424-6 Feng Shui at Home             12
ISBN-0-8021-4612-1 Baseball Heroes               15
```

Para expresar consultas más complejas que hagan referencia a funciones de tabla de SQL en subselecciones o que requieran diversas referencias de tabla en la cláusula FROM, se pueden utilizar expresiones de tabla comunes. El uso de una expresión de tabla común es una forma práctica de identificar la función de tabla de SQL en la selección más remota y de asegurarse de que la función de tabla que modifica datos de SQL sea la última referencia de tabla en la cláusula FROM.

Ejemplo 4: Invocación de una función de tabla de SQL que modifica datos de SQL correlacionada con otra referencia de tabla en una expresión de tabla común:

Este ejemplo amplía el ejemplo 3 devolviendo el precio de unidad y el valor de inventario total de los elementos en existencia actualizados. El valor de inventario total se calcula multiplicando las nuevas cantidades de estos elementos por el precio extraído de una tabla de lista de precios, `priceList`.

```
WITH newInv(itemNo, quantity) AS
  (SELECT id, TF.quantity
     FROM (VALUES ('ISBN-0-8021-3424-6', 5),
                  ('ISBN-0-8021-4612-1', 10)) AS newItem(id, q),
           TABLE(updateInv(newItem.id, newItem.q)) AS TF)
SELECT itemNo, quantity, unitPrice, (quantity * unitPrice) as TotalInvValue
  FROM newInv, priceList
    WHERE itemNo = priceList.itemID


ITEMNO             QUANTITY    UNITPRICE TOTALINVVALUE
------------------ ----------- --------- -----------------
ISBN-0-8021-3424-6          12     10.00            120.00
ISBN-0-8021-4612-1          15     20.00            300.70
```

Tareas relacionadas:

- "Auditoría mediante funciones de tabla de SQL" en la página 90

Información relacionada:

- "Sentencia CREATE FUNCTION (Escalar de SQL, tabla o fila)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencias de SQL que se permiten en rutinas" en la publicación *Consulta de SQL, Volumen 1*
- "Sentencias de SQL soportadas" en la publicación *Consulta de SQL, Volumen 2*

Auditoría mediante funciones de tabla de SQL

Los administradores de bases de datos interesados en supervisar accesos y modificaciones de datos de tabla efectuados por los usuarios de las bases de datos,

pueden comprobar las transacciones sobre una tabla creando y utilizando funciones de tabla de SQL que modifican datos de SQL.

Cualquier función de tabla que encapsula sentencias de SQL que realizan una tarea empresarial, como, por ejemplo, actualizar la información personal de un empleado, puede incluir adicionalmente sentencias de SQL que registren, en una tabla por separado, detalles sobre los accesos o modificaciones de tabla efectuados por el usuario que ha invocado la función. Una función de tabla de SQL se puede escribir incluso para que devuelva un conjunto de resultados de filas de tabla a las que se ha accedido o que se han modificado en el cuerpo de la función de tabla. Las filas del conjunto de resultados devuelto se pueden insertar y almacenar en una tabla por separado como histórico de los cambios efectuados en la tabla.

Requisitos previos:

Si desea conocer la lista de los privilegios necesarios para crear y registrar una función de tabla de SQL, vea las sentencias siguientes:

- Sentencia CREATE FUNCTION (escalar de SQL, tabla o fila)

El definidor de la función de tabla de SQL también debe tener autorización para ejecutar las sentencias de SQL encapsuladas en el cuerpo de dicha función. Consulte la lista de los privilegios necesarios para cada sentencia de SQL encapsulada. Para otorgar los privilegios INSERT, UPDATE y DELETE sobre una tabla a un usuario, vea la sentencia siguiente:

- Sentencia GRANT (privilegios de tabla, vista o apodo)

Las tablas a las que acceda la función de tabla de SQL deben existir antes de la invocación de la función de tabla de SQL.

Ejemplo 1: Auditoría de accesos de datos de tabla mediante una función de tabla de SQL:

Esta función accede a los datos de salario de todos los empleados de un departamento especificado por el argumento de entrada `deptno`. Asimismo, registra en una tabla de auditoría, denominada `audit_table`, el ID de usuario que ha invocado la función, el nombre de la tabla leída, una descripción de la información a la que se ha accedido y la hora actual. Observe que la función de tabla se crea con las palabras clave MODIFIES SQL DATA dado que contiene una sentencia INSERT que modifica datos de SQL.

```
CREATE FUNCTION sal_by_dept (deptno CHAR(3))
  RETURNS TABLE (lastname VARCHAR(10),
                 firstname VARCHAR(10),
                 salary INTEGER)
  LANGUAGE SQL
  MODIFIES SQL DATA
  NO EXTERNAL ACTION
  NOT DETERMINISTIC
  BEGIN ATOMIC
    INSERT INTO audit_table(user, table, action, time)
      VALUES (USER,
              'EMPLOYEE',
              'Read employee salaries in department: ' || deptno,
              CURRENT_TIMESTAMP);
    RETURN
      SELECT lastname, firstname, salary
        FROM employee as E
          WHERE E.dept = deptno;
  END
```

Ejemplo 2: Auditoría de actualizaciones en datos de tabla mediante una función de tabla de SQL:

Esta función actualiza el salario de un empleado especificado por updEmpNum, con la cantidad especificada por amount, y también registra en una tabla de auditoría, denominada audit_table, el usuario que ha invocado la rutina, el nombre de la tabla que se ha modificado y el tipo de modificación efectuada por el usuario. Se utiliza una sentencia SELECT que hace referencia a una sentencia de cambio de datos (aquí, una sentencia UPDATE) en la cláusula FROM para devolver los valores de fila actualizados. Observe que la función de tabla se crea con las palabras clave MODIFIES SQL DATA porque contiene tanto una sentencia INSERT como una sentencia SELECT que hace referencia a la sentencia de cambio de datos, UPDATE.

```
CREATE FUNCTION update_salary(updEmpNum CHAR(4), amount INTEGER)
  RETURNS TABLE (emp_lastname VARCHAR(10),
                 emp_firstname VARCHAR(10),
                 newSalary INTEGER)
  LANGUAGE SQL
  MODIFIES SQL DATA
  NO EXTERNAL ACTION
  NOT DETERMINISTIC
  BEGIN ATOMIC
    INSERT INTO audit_table(user, table, action, time)
    VALUES (USER,
            'EMPLOYEE',
            'Update emp salary. Values: '
               || updEmpNum || ' ' || char(amount),
            CURRENT_TIMESTAMP);
    RETURN
      SELECT lastname, firstname, salary
        FROM FINAL TABLE(UPDATE employee
                         SET salary = salary + amount
                         WHERE employee.empnum = updEmpNum);
  END
```

Ejemplo 3: Invocación de una función de tabla de SQL utilizada para la auditoría de transacciones:

A continuación, se muestra cómo un usuario puede invocar la rutina para actualizar el salario de un empleado con 500 yens:

```
SELECT emp_lastname, emp_firstname, newsalary
  FROM TABLE(update_salary(CHAR('1136'), 500)) AS T
```

Se devuelve un conjunto de resultados con el apellido, el nombre y el nuevo salario del empleado. El invocador de la función no sabrá que se ha efectuado el registro de auditoría.

```
EMP_LASTNAME EMP_FIRSTNAME NEWSALARY
------------ ------------- -----------
JONES        GWYNETH             90500
```

La tabla de auditoría incluiría un nuevo registro como el siguiente:

```
USER     TABLE      ACTION                              TIME
-------- ---------- ----------------------------------  --------------------------
MBROOKS  EMPLOYEE   Update emp salary. Values: 1136 500  2003-07-24-21.01.38.459255
```

Ejemplo 4: Recuperación de filas modificadas dentro del cuerpo de una función de tabla de SQL:

Esta función actualiza el salario de un empleado, especificado por un número de empleado EMPNUM, con una cantidad especificada por `amount`, y devuelve los valores originales de la fila o filas modificadas al llamante. Este ejemplo utiliza una sentencia SELECT que hace referencia a una sentencia de cambio de datos en la cláusula FROM. La especificación de OLD TABLE dentro de la cláusula FROM de esta sentencia indica la devolución de los datos de fila originales de la tabla `employee` que era el destino de la sentencia UPDATE. Si se utilizase FINAL TABLE, en lugar de OLD TABLE, indicaría la devolución de los valores de fila posteriores a la actualización de la tabla employee.

```
CREATE FUNCTION update_salary (updEmpNum CHAR(4), amount DOUBLE)
  RETURNS TABLE (empnum CHAR(4),
                 emp_lastname  VARCHAR(10),
                 emp_firstname VARCHAR(10),
                 dept CHAR(4),
                 newsalary integer)
  LANGUAGE SQL
  MODIFIES SQL DATA
  NO EXTERNAL ACTION
  DETERMINISTIC
  BEGIN ATOMIC
    RETURN
      SELECT empnum, lastname, firstname, dept, salary
        FROM OLD TABLE(UPDATE employee
                               SET salary = salary + amount
                                 WHERE employee.empnum = updEmpNum);
  END
```

Conceptos relacionados:

- "Funciones de tabla de SQL que modifican datos de SQL" en la página 88

Información relacionada:

- "Sentencia CREATE FUNCTION (Escalar de SQL, tabla o fila)" en la publicación *Consulta de SQL, Volumen 2*

Capítulo 4. Rutinas externas

Estilos de parámetros para rutinas externas 95
Sintaxis para pasar argumentos a rutinas escritas en C/C++, OLE o COBOL 97
SQL en rutinas externas 110
Efecto de la opción de vinculación DYNAMICRULES en SQL dinámico 113
Rutinas de ejecución en el lenguaje común .NET 116
| Rutinas de ejecución en el lenguaje común (CLR) 116
| Creación de rutinas CLR. 117
Tipos de datos de SQL soportados para el Proveedor de datos DB2 .NET 120
| Parámetros de las rutinas CLR 121
| Devolución de conjuntos de resultados desde procedimientos CLR 125
| Restricciones de las rutinas CLR 126
| Errores relacionados con rutinas CLR 128
| Ejemplos de procedimientos CLR en C# . . . 130
Ejemplos de procedimientos CLR en Visual Basic 142
Ejemplos de funciones CLR definidas por el usuario en C# 152
Ejemplos de funciones CLR definidas por el usuario en Visual Basic 158
Rutinas C/C++. 164
Rutinas C/C++. 164
Archivo de inclusión para rutinas C/C++ (sqludf.h). 168
Tipos de datos de SQL soportados en C/C++ 168
Manejo de tipos de datos de SQL en rutinas C/C++ 171
Variables gráficas del lenguaje principal en rutinas C/C++ 179
Decoración de tipos de C++ 179
Rutinas Java. 181
Rutinas Java. 181
Tipos de datos de SQL soportados en Java . . 185
Dónde se deben colocar las clases de Java . . . 186
Actualización de rutinas Java (procedimientos almacenados, UDF y métodos) para la ejecución. 187
Administración de los archivos JAR en el servidor de bases de datos 187
| Contextos de conexión en rutinas SQLJ. 189
Depuración de procedimientos almacenados en Java 189
Depuración de procedimientos almacenados de Java 189
Preparación para depurar procedimientos almacenados Java 190
Invocación del programa de depuración . . 191
Cómo llenar la tabla de depuración 192
Tabla de depuración de Java DB2DBG.ROUTINE_DEBUG 193
Rutinas de automatización de OLE 194
Diseño de rutinas de automatización de OLE 194
Creación de rutinas de automatización de OLE 195
Consideraciones sobre instancias de objetos y el área reutilizable y las rutinas de OLE 196
Tipos de datos de SQL soportados en la automatización de OLE 197
Rutinas de automatización de OLE en BASIC y C++ 198
Funciones de tabla de OLE DB definidas por el usuario 201
Funciones de tabla de OLE DB definidas por el usuario 201
Creación de una UDF para tablas OLE DB . . 203
Nombres de conjunto de filas completamente calificados 205
Tipos de datos de SQL soportados en OLE DB 206

Las rutinas externas se pueden escribir en los lenguajes de programación siguientes: C, C++, Java y OLE. Además de hacerlo en estos lenguajes, también es posible escribir procedimientos almacenados en COBOL.

A fin de construir una rutina externa, es necesario instalar y configurar los compiladores/kits de desarrollador soportados en el servidor de bases de datos según el lenguaje de la rutina. Para poder invocar las rutinas externas, antes se deben construir y registrar.

Estilos de parámetros para rutinas externas

Cada rutina se tiene que ajustar a un convenio determinado para el intercambio de parámetros. Estos convenios se conocen como *estilos de parámetros*. Se asigna un estilo de parámetros determinado a una rutina durante el registro de la misma, mediante la cláusula PARAMETER STYLE (ESTILO DE PARÁMETROS). A continuación se muestran los estilos de parámetros disponibles y los atributos de los mismos.

Tabla 1. Estilos de parámetros

Estilo de parámetros	Lenguaje soportado	Tipo de rutina soportado	Descripción
SQL [1]	• C/C++ • OLE • Lenguajes de ejecución en el lenguaje común .NET • COBOL [2]	• UDF • procedim. almacenados • métodos	Además de los parámetros que se pasan durante la invocación, se pasan a la rutina los argumentos siguientes por este orden: • Un indicador nulo para cada parámetro o resultado declarado en la sentencia CREATE. • El SQLSTATE que se debe devolver a DB2®. • El nombre calificado de la rutina. • El nombre específico de la rutina. • La serie de diagnóstico de SQL que se debe devolver a DB2. Según las opciones especificadas en la sentencia CREATE y el tipo de rutina, se pueden pasar a la rutina los argumentos siguientes por este orden: • Un almacenamiento intermedio para el área reutilizable. • El tipo de llamada de la rutina. • La estructura dbinfo (contiene información acerca de la base de datos).
DB2SQL [1]	• C/C++ • OLE • Lenguajes de ejecución en el lenguaje común .NET • COBOL	• procedim. almacenados	Además de los parámetros que se pasan durante la invocación, se pasan al procedimiento almacenado los argumentos siguientes por este orden: • Un vector que contiene un indicador nulo para cada parámetro de la sentencia CALL. • El SQLSTATE que se debe devolver a DB2. • El nombre calificado del procedimiento almacenado. • El nombre específico del procedimiento almacenado. • La serie de diagnóstico de SQL que se debe devolver a DB2. Si se especifica la cláusula DBINFO en la sentencia CREATE PROCEDURE, se pasa al procedimiento almacenado una estructura dbinfo (contiene información acerca de la base de datos).
JAVA	• Java™	• UDF • procedim. almacenados	Las rutinas con PARAMETER STYLE JAVA utilizan un convenio de pase de parámetros que cumple con la especificación del lenguaje Java y las rutinas de SQLJ. Para los procedimientos almacenados, los parámetros INOUT y OUT se pasarán como matrices de entrada única para facilitar la devolución de valores. Además de los parámetros IN, OUT e INOUT, las signaturas de método Java de los procedimientos almacenados incluyen un parámetro del tipo ResultSet[] para cada conjunto de resultados especificado en la cláusula DYNAMIC RESULT SETS de la sentencia CREATE PROCEDURE. Para las UDF y los métodos con PARAMETER STYLE JAVA, no se pasan más argumentos que los especificados en la invocación de la rutina.
DB2GENERAL	• Java	• UDF • procedim. almacenados • métodos	Este tipo de rutina utilizará un convenio de pase de parámetros definido para su uso con los métodos Java. A no ser que se desarrollen UDF de tabla o UDF con áreas reutilizables o que se tenga que acceder a la estructura dbinfo, es recomendable utilizar PARAMETER STYLE JAVA. Para las rutinas con PARAMETER STYLE DB2GENERAL, no se pasa ningún argumento adicional aparte de los especificados en la invocación de la rutina.
GENERAL	• C/C++ • Lenguajes de ejecución en el lenguaje común .NET • COBOL	• procedim. almacenados	Un procedimiento almacenado con PARAMETER STYLE GENERAL recibe parámetros de la sentencia CALL en la aplicación o rutina que realiza la invocación. Si se especifica la cláusula DBINFO en la sentencia CREATE PROCEDURE, se pasa al procedimiento almacenado una estructura dbinfo (contiene información acerca de la base de datos). GENERAL es el equivalente de los procedimientos almacenados SIMPLE en DB2 Universal Database para z/OS y OS/390.

Tabla 1. Estilos de parámetros (continuación)

Estilo de parámetros	Lenguaje soportado	Tipo de rutina soportado	Descripción
GENERAL WITH NULLS	• C/C++ • Lenguajes de ejecución en el lenguaje común .NET • COBOL	• procedim. almacenados	Un procedimiento almacenado con PARAMETER STYLE GENERAL WITH NULLS recibe parámetros de la sentencia CALL en la aplicación o rutina que realiza la invocación. También se incluye un vector que contiene un indicador nulo para cada parámetro de la sentencia CALL. Si se especifica la cláusula DBINFO en la sentencia CREATE PROCEDURE, se pasa al procedimiento almacenado una estructura dbinfo (contiene información acerca de la base de datos). GENERAL WITH NULLS es el equivalente de los procedimientos almacenados SIMPLE WITH NULLS en DB2 Universal Database para z/OS y OS/390.

Nota:

1. Para las UDF y los métodos, PARAMETER STYLE SQL es equivalente a PARAMETER STYLE DB2SQL.
2. COBOL sólo se puede utilizar para desarrollar procedimientos almacenados.
3. Los métodos de ejecución en el lenguaje común .NET no están soportados.

Conceptos relacionados:

- "Rutinas DB2GENERAL" en la página 357
- "Rutinas Java" en la página 181

Información relacionada:

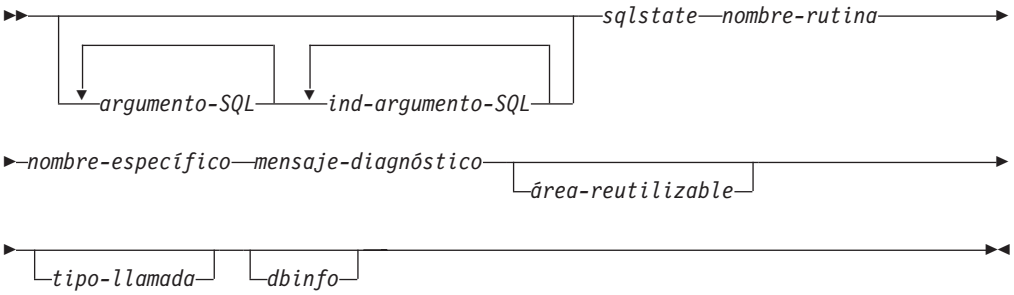
- "Sentencia CREATE FUNCTION" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE PROCEDURE" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE TYPE (Estructurado)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE METHOD" en la publicación *Consulta de SQL, Volumen 2*
- "Sintaxis para pasar argumentos a rutinas escritas en C/C++, OLE o COBOL" en la página 97

Sintaxis para pasar argumentos a rutinas escritas en C/C++, OLE o COBOL

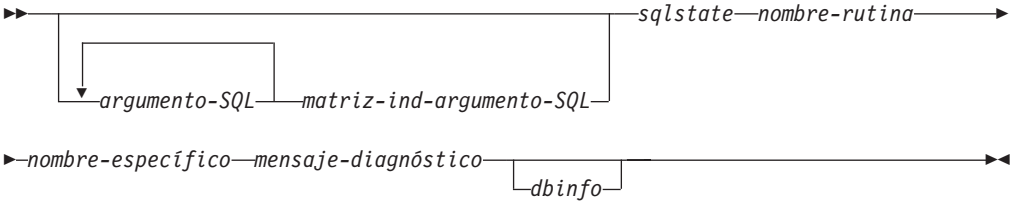
Además de los argumentos de SQL que se especifican en la referencia DML para una rutina, DB2 pasa argumentos adicionales al cuerpo de la rutina externa. La naturaleza y el orden de estos argumentos lo determina el estilo de parámetros con que se ha registrado la rutina. Para asegurarse de que se intercambie correctamente la información entre los invocadores y el cuerpo de la rutina, se tiene que cerciorar de que la rutina acepte los argumentos en el orden en que se le pasen, según el estilo de parámetros que se utilice. El archivo de inclusión sqludf le puede servir para el manejo y uso de estos argumentos.

Los estilos de parámetros siguientes sólo son aplicables a las rutinas LANGUAGE C, LANGUAGE OLE y LANGUAGE COBOL.

Rutinas PARAMETER STYLE SQL

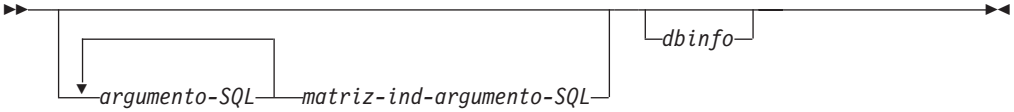


Procedimientos PARAMETER STYLE DB2SQL



Procedimientos PARAMETER STYLE GENERAL

Procedimientos PARAMETER STYLE GENERAL WITH NULLS



Nota: Para las UDF y los métodos, PARAMETER STYLE SQL es equivalente a PARAMETER STYLE DB2SQL.

Los argumentos para los estilos de parámetros anteriores se describen del modo siguiente:

argumento-SQL...
Cada *argumento-SQL* representa un valor de entrada o salida definido al crear la rutina. La lista de argumentos se determina del modo siguiente:

- Para una función escalar, un argumento para cada parámetro de entrada para la función, seguidos de un *argumento-SQL* para el resultado de la función.
- Para una función de tabla, un argumento para cada parámetro de entrada para la función, seguidos de un *argumento-SQL* para cada columna de la tabla de resultados de la función.

- Para un método, un *argumento-SQL* para el tipo sujeto del método, luego un argumento para cada parámetro de entrada para el método, seguidos de un *argumento-SQL* para el resultado del método.
- Para un procedimiento almacenado, un *argumento-SQL* para cada parámetro del procedimiento almacenado.

Cada uno de los *argumentos-SQL* se utiliza del modo siguiente:

- Parámetro de entrada de una función o método, tipo sujeto de un método o parámetro IN de un procedimiento almacenado

  DB2 establece este argumento antes de llamar a la rutina. El valor de cada uno de estos argumentos se toma de la expresión especificada en la invocación de la rutina. Se expresa en el tipo de datos de la definición del parámetro correspondiente en la sentencia CREATE.
- Resultado de una función o método, o parámetro OUT de un procedimiento almacenado

  La rutina establece este argumento antes de volver a DB2. DB2 asigna el almacenamiento intermedio y pasa la dirección del mismo a la rutina. La rutina coloca el valor del resultado en el almacenamiento intermedio. DB2 asigna un espacio de almacenamiento intermedio suficiente para que contenga el valor expresado en el tipo de datos. En los tipos de caracteres y LOB, esto significa que se asigna el tamaño máximo definido en la sentencia de creación.

  Para los métodos y funciones escalares, el tipo de datos del resultado se define en la cláusula CAST FROM, si está presente, o en la cláusula RETURNS, si no hay ninguna cláusula CAST FROM presente.

  Para las funciones de tabla, DB2 define una optimización del rendimiento si no se tienen que devolver a DB2 todas las columnas definidas. Si escribe una UDF que se aproveche de esta característica, ésta sólo devolverá las columnas que necesite la sentencia que hace referencia a la función de tabla. Por ejemplo, piense en una sentencia CREATE FUNCTION para una función de tabla definida con 100 columnas de resultado. Si una sentencia determinada que hace referencia a la función sólo está interesada en dos de ellas, esta optimización permite que la UDF sólo devuelva estas dos columnas de cada fila, y no invierta tiempo en las otras 98 columnas. Para obtener más información sobre la optimización mencionada, consulte el argumento dbinfo más adelante.

  Por cada valor devuelto, la rutina no debe devolver más bytes de los necesarios para el tipo de datos y la longitud del resultado. Los valores máximos se definen durante la creación de la entrada de catálogo de la rutina. Una sobregrabación por parte de la rutina puede ocasionar resultados imprevisibles o una terminación anómala.
- Parámetro INOUT de un procedimiento almacenado

  Este argumento se comporta como los parámetros IN y OUT y, por consiguiente, sigue los dos conjuntos de reglas indicados anteriormente. DB2 establecerá el argumento antes de llamar al procedimiento almacenado. El almacenamiento intermedio asignado por DB2 para el argumento es lo suficientemente grande como para contener el tamaño máximo del tipo de datos del parámetro definido en la sentencia CREATE PROCEDURE. Por ejemplo, un parámetro INOUT de tipo CHAR puede tener un varchar de 10 bytes que entre en el procedimiento almacenado y un varchar de 100 bytes que salga del procedimiento almacenado. El procedimiento almacenado establece el almacenamiento intermedio antes de volver a DB2.

DB2 alinea los datos para *argumento-SQL* en función del tipo de datos y del sistema operativo del servidor, también conocido como plataforma.

ind-argumento-SQL...
Hay un *ind-argumento-SQL* para cada *argumento-SQL* que se pase a la rutina. El *ind-argumento-SQL* número *n* corresponde al *argumento-SQL* número *n* e indica si el *argumento-SQL* tiene un valor o es nulo (NULL).

Cada uno de los *ind-argumento-SQL* se utiliza del modo siguiente:

- Parámetro de entrada de una función o método, tipo sujeto de un método o parámetro IN de un procedimiento almacenado

  DB2 establece este argumento antes de llamar a la rutina. Contiene uno de los valores siguientes:
  0 El argumento está presente y no es NULL.
  -1 El argumento está presente y su valor es NULL.

  Si la rutina se ha definido con RETURNS NULL ON NULL INPUT, el cuerpo de la rutina no tiene necesidad de comprobar un valor NULL. Sin embargo, si se ha definido con CALLED ON NULL INPUT, cualquier argumento puede ser NULL y la rutina debe comprobar el *ind-argumento-SQL* antes de utilizar el *argumento-SQL* correspondiente.
- Resultado de una función o método, o parámetro OUT de un procedimiento almacenado

  La rutina establece este argumento antes de volver a DB2. La rutina utiliza este argumento para señalar si el valor del resultado en concreto es NULL:
  0 El resultado no es NULL.
  -1 El resultado es el valor NULL.

  Aunque la rutina se haya definido con RETURNS NULL ON NULL INPUT, el cuerpo de la rutina tiene que establecer el *ind-argumento-SQL* del resultado. Por ejemplo, una función de división puede establecer el resultado en nulo cuando el denominador sea cero.

  Para los métodos y funciones escalares, DB2 trata un resultado NULL como un error aritmético si se dan las circunstancias siguientes:
  – El parámetro de configuración de base de datos *dft_sqlmathwarn* es YES
  – Uno de los argumentos de entrada es nulo debido a un error aritmético

  También sucede así si se define la función con la opción RETURNS NULL ON NULL INPUT

  Para las funciones de tabla, si la UDF se aprovecha de la optimización utilizando la lista de columnas del resultado, sólo es necesario establecer los indicadores correspondientes a las columnas requeridas.
- Parámetro INOUT de un procedimiento almacenado

  Este argumento se comporta como los parámetros IN y OUT y, por consiguiente, sigue los dos conjuntos de reglas indicados anteriormente. DB2 establecerá el argumento antes de llamar al procedimiento almacenado. El procedimiento almacenado establece el *ind-argumento-SQL* antes de volver a DB2.

Cada *ind-argumento-SQL* toma la forma de un valor SMALLINT. DB2 alinea los datos para *ind-argumento-SQL* en función del tipo de datos y del sistema operativo del servidor.

matriz-ind-argumento-SQL
En la *matriz-ind-argumento-SQL*, existe un elemento para cada argumento-SQL pasado al procedimiento almacenado. El elemento número *n* de la *matriz-ind-argumento-SQL* corresponde al argumento-SQL número *n* e indica si el *argumento-SQL* tiene un valor o es NULL.

Cada elemento de la *matriz-ind-argumento-SQL* se utiliza del modo siguiente:

- Parámetro IN de un procedimiento almacenado

  DB2 establece este elemento antes de llamar a la rutina. Contiene uno de los valores siguientes:
  0 El argumento está presente y no es NULL.
  -1 El argumento está presente y su valor es NULL.

  Si el procedimiento almacenado se ha definido con RETURNS NULL ON NULL INPUT, el cuerpo del procedimiento almacenado no tiene necesidad de comprobar un valor NULL. Sin embargo, si se ha definido con CALLED ON NULL INPUT, cualquier argumento puede ser NULL y el procedimiento almacenado debe comprobar el *ind-argumento-SQL* antes de utilizar el *argumento-SQL* correspondiente.
- Parámetro OUT de un procedimiento almacenado

  La rutina establece este elemento antes de volver a DB2. La rutina utiliza este argumento para señalar si el valor del resultado en concreto es NULL:
  0 o positivo
  El resultado no es NULL.
  negativo
  El resultado es el valor NULL.
- Parámetro INOUT de un procedimiento almacenado

  Este elemento se comporta como los parámetros IN y OUT y, por consiguiente, sigue los dos conjuntos de reglas indicados anteriormente. DB2 establecerá el argumento antes de llamar al procedimiento almacenado. El procedimiento almacenado establece el elemento de *matriz-ind-argumento-SQL* antes de volver a DB2.

Cada elemento de *matriz-ind-argumento-SQL* toma la forma de un valor SMALLINT. DB2 alinea los datos para *matriz-ind-argumento-SQL* en función del tipo de datos y del sistema operativo del servidor.

sqlstate La rutina establece este argumento antes de volver a DB2. Ésta lo puede utilizar para señalar condiciones de aviso o error. La rutina puede establecer este argumento en cualquier valor. El valor '00000' significa que no se ha detectado ninguna condición de aviso ni de error. Los valores que empiezan por '01' son condiciones de aviso. Los valores que empiezan por cualquier cosa distinta de '00' y '01' son condiciones de error. Cuando se llama a la rutina, el argumento contiene el valor '00000'.

Para las condiciones de error, la rutina devuelve un SQLCODE de -443. Para las condiciones de aviso, devuelve un SQLCODE de +462. Si el SQLSTATE es 38001 ó 38502, el SQLCODE es -487.

El *sqlstate* toma la forma de un valor CHAR(5). DB2 alinea los datos para *sqlstate* en función del tipo de datos y del sistema operativo del servidor.

nombre-rutina
DB2 establece este argumento antes de llamar a la rutina. Se trata del nombre de función calificado que se pasa de DB2 a la rutina

El formato del *nombre-rutina* que se pasa es el siguiente:

```
esquema.rutina
```

Las distintas partes se separan mediante un punto. Éstos son dos ejemplos:

```
PABLO.BLOOP   WILLIE.FINDSTRING
```

Este formato permite utilizar el mismo cuerpo de rutina para varias rutinas externas, y sigue diferenciando entre las rutinas cuando se las invoca.

Nota: Aunque es posible incluir un punto en nombres de objeto y nombres de esquema, no es aconsejable. Por ejemplo, si una función `ROTATE` se encuentra en un esquema `OBJ.OP`, el nombre de rutina que se pasa a la función es `OBJ.OP.ROTATE`, y no resulta obvio si el nombre de esquema es `OBJ` u `OBJ.OP`.

El *nombre-rutina* adopta el formato de un valor VARCHAR(257). DB2 alinea los datos para *nombre-rutina* en función del tipo de datos y del sistema operativo del servidor.

nombre-específico
DB2 establece este argumento antes de llamar a la rutina. Se trata del nombre específico de la rutina que se pasa de DB2 a la rutina.

Éstos son dos ejemplos:

```
WILLIE_FIND_FEB99 SQL9904281052440430
```

El usuario proporciona este primer valor en su sentencia CREATE. El segundo valor, si el usuario no especifica ninguno, lo genera DB2 a partir de la indicación de la hora actual.

Al igual que con el argumento *nombre-rutina*, la razón para pasar este valor consiste en brindar a la rutina un medio para distinguir exactamente qué rutina concreta lo está invocando.

El *nombre-específico* toma la forma de un valor VARCHAR(18). DB2 alinea los datos para *nombre-específico* en función del tipo de datos y del sistema operativo del servidor.

mensaje-diagnóstico
La rutina establece este argumento antes de volver a DB2. La rutina puede utilizar este argumento para insertar texto de mensaje en un mensaje de DB2.

Cuando la rutina devuelve un error o un aviso, utilizando el argumento *sqlstate* descrito anteriormente, puede incluir aquí información descriptiva. DB2 incluye esta información como señal en su mensaje.

DB2 establece el primer carácter en nulo antes de llamar a la rutina. Al volver, trata la serie como una serie C terminada en nulo. Esta serie se incluirá en la SQLCA como señal de la condición de error. Como mínimo la primera parte de esta serie aparecerá en la SQLCA o en el mensaje del CLP de DB2. No obstante, el número real de caracteres que aparecerán depende de las longitudes de las otras señales, puesto que DB2 trunca las señales para que se ajusten al límite impuesto por la SQLCA respecto a la longitud total de las señales. Evite utilizar X'FF' en el texto, ya que este carácter se utiliza para delimitar señales en la SQLCA.

La rutina no debe devolver más texto del que quepa en el almacenamiento intermedio de VARCHAR(70) que se le pasa. Una sobregrabación por parte de la rutina puede ocasionar resultados imprevisibles o una terminación anómala.

DB2 supone que las señales de mensajes devueltas a DB2 por la rutina están en la misma página de códigos que la rutina. La rutina se debe asegurar de que es así. Si se utiliza el subconjunto ASCII invariable de 7 bits, la rutina puede devolver las señales de mensajes en cualquier página de códigos.

El *mensaje-diagnóstico* toma la forma de un valor VARCHAR(70). DB2 alinea los datos para *mensaje-diagnóstico* en función del tipo de datos y del sistema operativo del servidor.

área-reutilizable

DB2 establece este argumento antes de invocar la UDF o al método. Sólo está presente para las funciones y los métodos en que se ha especificado la palabra clave SCRATCHPAD durante el registro. Este argumento es una estructura, exactamente igual que la estructura utilizada para pasar un valor de cualquiera de los tipos de datos LOB, con los elementos siguientes:

- Un INTEGER que contiene la longitud del área reutilizable. Si se cambia la longitud del área reutilizable, se producirá un SQLCODE -450 (SQLSTATE 39501)
- El área reutilizable real inicializada completamente con ceros binarios del modo siguiente:
  - Para los métodos y las funciones escalares, se inicializa antes de la primera llamada y DB2 no la suele examinar ni modificar después.
  - Para las funciones de tabla, el área reutilizable se inicializa antes de la primera (FIRST) llamada a la UDF si se especifica FINAL CALL en la sentencia CREATE FUNCTION. A partir de esta llamada, el contenido del área reutilizable está totalmente bajo el control de la función de tabla. Si se ha especificado, o asumido por omisión, NO FINAL CALL para una función de tabla, el área reutilizable se inicializa para cada llamada OPEN, y el contenido de la misma está totalmente bajo el control de la función de tabla entre llamadas OPEN. (Esto puede ser muy importante para una función de tabla utilizada en una unión o en una subconsulta. Si es necesario mantener el contenido del área reutilizable a lo largo de las llamadas OPEN, se debe especificar FINAL CALL en la sentencia CREATE FUNCTION. Si se especifica FINAL CALL, además de las llamadas OPEN, FETCH y CLOSE normales, la función de tabla recibirá también llamadas FIRST y FINAL, con el objetivo de mantener el área reutilizable y liberar recursos.)

El área reutilizable se puede correlacionar en la rutina utilizando el mismo tipo que un CLOB o un BLOB, puesto que el argumento que se pasa tiene la misma estructura.

Asegúrese de que el código de la rutina no realiza cambios fuera del almacenamiento intermedio del área reutilizable. Una sobregrabación por parte de la rutina puede causar resultados imprevisibles o una terminación anómala y puede que DB2 experimente una anomalía no leve.

Si en una subconsulta se hace referencia a un método o una UDF escalar que emplea un área reutilizable, es posible que DB2 decida renovar el área reutilizable entre invocaciones de la subconsulta. Esta renovación se

produce después de realizar una llamada final, en caso de que se haya especificado FINAL CALL para la UDF.

DB2 inicializa el área reutilizable de forma que el campo de datos esté alineado para el almacenamiento de cualquier tipo de datos. Esto puede hacer que la estructura entera del área reutilizable, incluido el campo de longitud, esté alineada incorrectamente.

tipo-llamada
DB2 establece este argumento, si está presente, antes de invocar la UDF o al método. Este argumento está presente para todas las funciones de tabla y para los métodos y funciones escalares en que se ha especificado FINAL CALL durante el registro.

A continuación, se indican todos los valores posibles actuales para *tipo-llamada*. La UDF o el método debe contener una sentencia de conmutación o de caso que pruebe explícitamente todos los valores esperados, en lugar de contener lógica del tipo "if A do AA, else if B do BB, else it must be C so do CC". Esto es así porque puede que en el futuro se añadan tipos de llamadas adicionales y, si no prueba explícitamente la condición C, tendrá problemas cuando se añadan las nuevas posibilidades.

Notas:

1. Para todos los valores de *tipo-llamada*, puede ser conveniente que la rutina establezca un valor de retorno de *sqlstate* y de *mensaje-diagnóstico*. Esta información no se repetirá en las descripciones siguientes de cada uno de los *tipos-llamada*. Para todas las llamadas, DB2 emprenderá la acción indicada descrita anteriormente para estos argumentos.
2. El archivo de inclusión `sqludf.h` está destinado a la utilización con rutinas. El archivo contiene definiciones simbólicas para los valores de *tipo-llamada* siguientes, que se escriben como constantes.

Para los métodos y funciones escalares, *tipo-llamada* contiene:

SQLUDF_FIRST_CALL (-1)
Ésta es la primera (FIRST) llamada a la rutina para esta sentencia. El *área reutilizable* (si la hay) se establece con ceros binarios cuando se llama a la rutina. Se pasan los valores de todos los argumentos y la rutina debe realizar las acciones puntuales de inicialización que sean necesarias. Además, una llamada FIRST a un método o una UDF escalar es como una llamada NORMAL, en el sentido en que se espera que desarrolle y devuelva una respuesta.

Nota: Si se especifica SCRATCHPAD pero no así FINAL CALL, la rutina no tendrá este argumento de *tipo-llamada* para identificar la llamada realmente primera. En lugar de esto, se tendrá que basar en el estado del área reutilizable, todo ceros.

SQLUDF_NORMAL_CALL (0)
Ésta es una llamada NORMAL. Se pasan todos los valores de entrada de SQL y se espera que la rutina desarrolle y devuelva el resultado. También puede que la rutina devuelva información de *sqlstate* y de *mensaje-diagnóstico*.

SQLUDF_FINAL_CALL (1)
Ésta es una llamada FINAL; es decir, no se pasa ningún valor de *argumento-SQL* ni de *ind-argumento-SQL*, y los

intentos de examinar estos valores pueden causar resultados imprevisibles. Si también se pasa un *área reutilizable*, ésta permanece inalterada desde la llamada anterior. Se espera que la rutina libere recursos en este punto.

SQLUDF_FINAL_CRA (255)
Ésta es una llamada FINAL, idéntica a la llamada FINAL descrita antes, con una característica adicional, a saber, que se crea para rutinas que están definidas como capaces de emitir SQL, y se crea en un momento en que la rutina no debe emitir SQL, a excepción de CLOSE cursor. (SQLCODE -396, SQLSTATE 38505) Por ejemplo, cuando DB2 está en medio de un proceso COMMIT, no puede tolerar nuevo SQL, y cualquier llamada FINAL emitida para una rutina en ese momento será una llamada 255 FINAL. Las rutinas que no estén definidas para contener ningún nivel de acceso a SQL nunca recibirán una llamada 255 FINAL, mientras que las rutinas que utilizan SQL pueden recibir cualquier tipo de llamada FINAL.

Liberación de recursos

De un método o una UDF escalar se espera que liberen los recursos que han necesitado, por ejemplo memoria. Si se especifica FINAL CALL para la rutina, esa llamada FINAL es el lugar natural en el que liberar recursos, siempre que también se especifique SCRATCHPAD y se utilice para hacer un seguimiento del recurso. Si no se especifica FINAL CALL, cualquier recurso adquirido se debe liberar en la misma llamada.

Para las funciones de tabla, *tipo-llamada* contiene:

SQLUDF_TF_FIRST (-2)
Ésta es la primera (FIRST) llamada, que sólo se produce si se ha especificado la palabra clave FINAL CALL para la UDF. El *área reutilizable* se establece con ceros binarios antes de esta llamada. Los valores de los argumentos se pasan a la función de tabla. La función de tabla puede adquirir memoria o realizar otra inicialización puntual de sólo recursos. No se trata de una llamada OPEN, la cual sigue a ésta. En una llamada FIRST, la función de tabla no debe devolver ningún dato a DB2, ya que DB2 ignora los datos.

SQLUDF_TF_OPEN (-1)
Ésta es la llamada OPEN. El *área reutilizable* se inicializará si se especifica NO FINAL CALL, pero no necesariamente en caso contrario. En OPEN se pasan los valores de todos los argumentos de SQL a la función de tabla. La función de tabla no debe devolver ningún dato a DB2 en la llamada OPEN.

SQLUDF_TF_FETCH (0)
Ésta es una llamada FETCH y DB2 espera que la función de tabla devuelva una fila que comprenda el conjunto de valores de retorno o una condición de fin de tabla, indicada por el valor '02000' de SQLSTATE. Si se pasa el *área reutilizable* a la UDF, al entrar ésta permanece inalterada respecto a la llamada anterior.

SQLUDF_TF_CLOSE (1)
Ésta es una llamada CLOSE para la función de tabla. Equilibra la llamada OPEN y se puede utilizar para realizar cualquier proceso CLOSE externo (por ejemplo, cerrar un archivo fuente) y para la liberación de recursos (especialmente para el caso de NO FINAL CALL).

En los casos que implican una unión o una subconsulta, se pueden repetir secuencias de llamadas OPEN/FETCH.../CLOSE dentro de la ejecución una sentencia, pero sólo habrá una llamada FIRST y una llamada FINAL. Las llamadas FIRST y FINAL sólo se producen si se especifica FINAL CALL para la función de tabla.

SQLUDF_TF_FINAL (2)
Ésta es una llamada FINAL, que sólo se produce si se ha especificado FINAL CALL para la función de tabla. Equilibra la llamada FIRST y sólo se produce una vez por ejecución de la sentencia. Está destinada a liberar recursos.

SQLUDF_TF_FINAL_CRA (255)
Ésta es una llamada FINAL, idéntica a la llamada FINAL descrita antes, con una característica adicional, digamos que está formada por UDF que están definidas como capaces de emitir SQL, y se ha creado en un momento en que la UDF no debe emitir SQL, a excepción de CLOSE cursor. (SQLCODE -396, SQLSTATE 38505) Por ejemplo, cuando DB2 está en medio de un proceso COMMIT, no puede tolerar nuevo SQL, y cualquier llamada FINAL emitida para una UDF en ese momento será una llamada 255 FINAL. Tenga en cuenta que las UDF que no están definidas para contener un nivel de acceso a SQL nunca recibirán una llamada 255 FINAL, mientras que las UDF que utilizan SQL pueden recibir cualquier tipo de llamada FINAL.

Liberación de recursos

Escriba las rutinas de forma que liberen los recursos que adquieran. Para las funciones de tabla, existen dos lugares naturales para esta liberación: la llamada CLOSE y la llamada FINAL. La llamada CLOSE equilibra cada llamada OPEN y se puede producir varias veces en la ejecución de una sentencia. La llamada FINAL sólo se produce si se especifica FINAL CALL para la UDF, y únicamente se produce una vez por sentencia.

Si puede aplicar un recurso en todas las secuencias de OPEN/FETCH/CLOSE de la UDF, escriba la UDF de forma que adquiera el recurso en la llamada FIRST y lo libere en la llamada FINAL. El área reutilizable es un lugar natural para hacer un seguimiento de este recurso. Para las funciones de tabla, si se especifica FINAL CALL, el área reutilizable sólo se inicializa antes de la llamada FIRST. Si no se especifica FINAL CALL, se reinicializa antes de cada llamada OPEN.

Si un recurso es específico para cada secuencia de OPEN/FETCH/CLOSE, escriba la UDF de forma que libere el recurso en la llamada CLOSE.

Nota: Cuando una función de tabla está en una subconsulta o unión, es muy posible que se produzcan varias apariciones de la secuencia de

OPEN/FETCH/CLOSE, según cómo decida organizar la ejecución de la sentencia el Optimizador de DB2.

El *tipo-llamada* toma la forma de un valor INTEGER. DB2 alinea los datos para *tipo-llamada* en función del tipo de datos y del sistema operativo del servidor.

dbinfo DB2 establece este argumento antes de llamar a la rutina. Sólo está presente si la sentencia CREATE para la rutina especifica la palabra clave DBINFO. El argumento es la estructura `sqludf_dbinfo` definida en el archivo de cabecera `sqludf.h`. Las variables de esta estructura que contienen nombres e identificadores pueden ser más largas que el valor más largo posible en este release de DB2, pero se definen de esta manera a efectos de compatibilidad con releases futuros. Puede utilizar la variable de longitud que complementa cada variable de nombre y de identificador para leer o extraer la porción de la variable que se utilice realmente. La estructura *dbinfo* contiene los elementos siguientes:

1. Longitud del nombre de base de datos (dbnamelen)

   Longitud del *nombre de base de datos* siguiente. Este campo es un entero corto sin signo.

2. Nombre de base de datos (dbname)

   Nombre de la base de datos conectada actualmente. Este campo es un identificador largo de 128 caracteres. El campo *longitud del nombre de base de datos* descrito anteriormente identifica la longitud real de este campo. No contiene un terminador nulo ni ningún relleno.

3. Longitud del ID de autorización de la aplicación (authidlen)

   Longitud del *ID de autorización de la aplicación* siguiente. Este campo es un entero corto sin signo.

4. ID de autorización de la aplicación (authid)

   ID de autorización de ejecución de la aplicación. Este campo es un identificador largo de 128 caracteres. No contiene un terminador nulo ni ningún relleno. El campo *longitud del ID de autorización de la aplicación* descrito anteriormente identifica la longitud real de este campo.

5. Páginas de códigos del entorno (codepg)

   Se trata de una unión de tres estructuras de 48 bytes; una es común para todos los productos de DB2 Universal Database (cdpg_db2), otra se utiliza en las rutinas escritas para versiones anteriores de DB2 Universal Database (cdpg_cs) y la última se utiliza en versiones anteriores de DB2 UDB para z/OS y OS/390 (cdpg_mvs). Con miras a la portabilidad, es recomendable que la estructura común, cdpg_db2, se utilice en todas las rutinas.

   La estructura cdgp_db2 está formada por una matriz (db2_ccsids_triplet) de tres conjuntos de información de página de códigos que representa los esquemas de codificación posibles de la base de datos del modo siguiente:

   a. Esquema de codificación ASCII. Tenga en cuenta que, para la compatibilidad con la versión anterior de DB2 Universal Database, si la base de datos es Unicode, la información correspondiente al esquema de codificación Unicode se colocará aquí, además de aparecer en el tercer elemento.
   b. Esquema de codificación EBCDIC
   c. Esquema de codificación Unicode

Después de la información de los esquemas de codificación, se encuentra el índice de matriz del esquema de codificación para la rutina (db2_encoding_scheme).

Cada elemento de la matriz está compuesto por tres campos:
- db2_sbcs. Página de códigos de un solo byte, un entero largo sin signo.
- db2_dbcs. Página de códigos de doble byte, un entero largo sin signo.
- db2_mixed. Página de códigos compuesta (también conocida como página de códigos mixta), un entero largo sin signo.

6. Longitud del nombre de esquema (tbschemalen)

   Longitud del *nombre del esquema* siguiente. Contiene 0 (cero) si no se pasa un nombre de tabla. Este campo es un entero corto sin signo.

7. Nombre del esquema (tbschema)

   Esquema del *nombre de la tabla* siguiente. Este campo es un identificador largo de 128 caracteres. No contiene un terminador nulo ni ningún relleno. El campo *longitud del nombre de esquema* descrito anteriormente identifica la longitud real de este campo.

8. Longitud del nombre de tabla (tbnamelen)

   Longitud del *nombre de la tabla* siguiente. Contiene 0 (cero) si no se pasa un nombre de tabla. Este campo es un entero corto sin signo.

9. Nombre de la tabla (tbname)

   Nombre de la tabla que se está actualizando o insertando. Este campo sólo se establece si la referencia a la rutina se encuentra a la derecha de una cláusula SET de una sentencia UPDATE o si es un elemento de la lista VALUES de una sentencia INSERT. Este campo es un identificador largo de 128 caracteres. No contiene un terminador nulo ni ningún relleno. El campo *longitud del nombre de tabla* descrito anteriormente identifica la longitud real de este campo. El campo *nombre del esquema* descrito anteriormente forma, junto con este campo, el nombre de tabla completamente calificado.

10. Longitud del nombre de columna (colnamelen)

    Longitud del *nombre de columna* siguiente. Contiene un 0 (cero) si no se pasa un nombre de columna. Este campo es un entero corto sin signo.

11. Nombre de la columna (colname)

    Bajo las mismas condiciones exactas que se aplican al nombre de la tabla, este campo contiene el nombre de la columna que se ha de actualizar o insertar; de lo contrario, es imprevisible. Este campo es un identificador largo de 128 caracteres. No contiene un terminador nulo ni ningún relleno. El campo *longitud del nombre de columna* descrito anteriormente identifica la longitud real de este campo.

12. Número de versión/release (ver_rel)

    Campo de 8 caracteres que identifica el producto y la versión, el release y el nivel de modificación del mismo, con el formato *pppvvrrm*, donde:
    - *ppp* identifica el producto, del modo siguiente:
      DSN DB2 Universal Database para z/OS u OS/390
      ARI SQL/DS o DB2 para VM o VSE
      QSQ DB2 Universal Database para iSeries
      SQL DB2 Universal Database
    - *vv* es un identificador de versión de dos dígitos.

- *rr* es un identificador de release de dos dígitos.
- *m* es un identificador de nivel de modificación de un dígito.

13. Campo reservado (resd0)

    Este campo es para usos futuros.

14. Plataforma (platform)

    Sistema operativo (plataforma) del servidor de aplicaciones, tal como se indica a continuación:

    SQLUDF_PLATFORM_AIX AIX
    SQLUDF_PLATFORM_HP HP-UX
    SQLUDF_PLATFORM_LINUX
    Linux
    SQLUDF_PLATFORM_MVS OS/390
    SQLUDF_PLATFORM_NT Windows NT, Windows 2000, Windows XP
    SQLUDF_PLATFORM_SUN Solaris Operating Environment
    SQLUDF_PLATFORM_WINDOWS95
    Windows 95, Windows 98, Windows Me
    SQLUDF_PLATFORM_UNKNOWN
    Plataforma o sistema operativo desconocido

    Para informarse sobre sistemas operativos adicionales no incluidos en la lista anterior, consulte el contenido del archivo `sqludf.h`.

15. Número de entradas de la lista de columnas de la función de tabla (numtfcol)

    Número de entradas distintas de cero que existen en la lista de columnas de la función de tabla especificada en el campo *lista de columnas de la función de tabla* siguiente.

16. Campo reservado (resd1)

    Este campo es para usos futuros.

17. ID de rutina del procedimiento almacenado que ha invocado la rutina actual (procid)

    El ID de rutina del procedimiento almacenado coincide con la columna ROUTINEID de SYSCAT.ROUTINES, que se puede utilizar para recuperar el nombre del procedimiento almacenado que realiza la invocación. Este campo es un entero con signo de 32 bits.

18. Campo reservado (resd2)

    Este campo es para usos futuros.

19. Lista de columnas de la función de tabla (tfcolumn)

    Si ésta es una función de tabla, este campo es un puntero a una matriz de enteros cortos que DB2 asigna dinámicamente. Si se trata de cualquier otro tipo de rutina, este puntero es nulo.

    Sólo se utiliza este campo para las funciones de tabla. Sólo son de interés las *n* primeras entradas, donde *n* se especifica en el campo *número de entradas de la lista de columnas de la función de tabla*, `numtfcol`. *n* puede equivaler a 0, y *n* es inferior o igual al número de columnas de resultado definidas para la función en la cláusula RETURNS TABLE(...) de la sentencia CREATE FUNCTION. Los valores corresponden a los números ordinales de las columnas que esta sentencia necesita de la función de tabla. Un valor de '1' significa la primera columna de resultado definida, '2' significa la segunda columna de resultado definida, y así sucesivamente, y los valores

pueden seguir cualquier orden. Tenga en cuenta que *n* puede ser igual a cero, es decir, que la variable `numtfcol` puede ser cero, para una sentencia parecida a `SELECT COUNT(*) FROM TABLE(TF(...)) AS QQ`, en que la consulta no necesita ningún valor de columna real.

Esta matriz representa una oportunidad para realizar una optimización. La UDF no tiene necesidad de devolver todos los valores de todas las columnas de resultado de la función de tabla, únicamente de aquéllas que sean necesarias en el contexto en concreto, y éstas son las columnas identificadas (por número) en la matriz. Dado que esta optimización puede complicar la lógica de la UDF para lograr beneficios en el rendimiento, la UDF puede elegir devolver todas las columnas definidas.

20. Identificador exclusivo de la aplicación (appl_id)

    Este campo es un puntero a una serie C terminada en nulo que identifica de forma exclusiva la conexión de la aplicación con DB2. DB2 lo genera durante la conexión.

    La serie tiene una longitud máxima de 32 caracteres y su formato exacto depende del tipo de conexión establecida entre el cliente y DB2. Generalmente, toma la forma siguiente:

    ```
    x.y.ts
    ```

    donde *x* e *y* varían según el tipo de conexión, pero *ts* es una indicación de la hora de 12 caracteres con formato AAMMDDHHMMSS, que DB2 ajusta potencialmente para asegurar su exclusividad.

    ```
    Ejemplo:  *LOCAL.db2inst.980707130144
    ```

21. Campo reservado (resd3)

    Este campo es para usos futuros.

Conceptos relacionados:

- "Estilos de parámetros para rutinas externas" en la página 95
- "Archivo de inclusión para rutinas C/C++ (sqludf.h)" en la página 168
- "Rutinas C/C++" en la página 164

Tareas relacionadas:

- "Escritura de rutinas" en la página 36

Información relacionada:

- "Sentencia CREATE FUNCTION" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE PROCEDURE" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE TYPE (Estructurado)" en la publicación *Consulta de SQL, Volumen 2*
- "appl_id - Application ID monitor element" en la publicación *System Monitor Guide and Reference*
- "Sentencia CREATE METHOD" en la publicación *Consulta de SQL, Volumen 2*

SQL en rutinas externas

Todas las rutinas escritas en un lenguaje de programación externo (como C, Visual Basic, C#, Java™ y otros) pueden contener SQL.

La sentencia CREATE para una rutina (procedimiento almacenado o UDF) o la sentencia CREATE TYPE, en el caso de un método, contiene una cláusula que define el nivel de acceso de SQL para la rutina o método. En base a la naturaleza del SQL incluido en la rutina, deberá elegir la cláusula pertinente.

NO SQL
La rutina no contiene SQL en absoluto.

CONTAINS SQL
Contiene SQL, pero no lee ni graba datos (por ejemplo: SET SPECIAL REGISTER).

READS SQL DATA
Contiene SQL que puede leer tablas (sentencias SELECT, VALUES), pero no modifica datos de tabla.

MODIFIES SQL DATA
Contiene SQL que actualiza tablas, ya sean tablas del usuario directamente (sentencias INSERT, UPDATE, DELETE) o tablas del catálogo de DB2® implícitamente (sentencias de DDL). Esta cláusula sólo es aplicable a procedimientos almacenados y funciones de tabla incorporadas al SQL.

DB2 validará, durante la ejecución, que una rutina no supere su nivel definido. Por ejemplo, si una rutina definida como CONTAINS SQL intenta realizar SELECT en una tabla, dará como resultado un error (SQLCODE -579, SQLSTATE 38004) porque intenta una lectura de datos de SQL. Observe también que las referencias de rutinas anidadas deben tener el mismo nivel de SQL, o más restrictivo, que contiene la referencia. Por ejemplo, las rutinas que modifican datos de SQL pueden invocar a rutinas que leen datos de SQL, pero las rutinas que sólo pueden leer datos de SQL, definidas con la cláusula READS SQL DATA, no pueden invocar a rutinas que modifican datos de SQL.

Una rutina ejecuta sentencias de SQL dentro del ámbito de conexión de base de datos de la aplicación que realiza la llamada. Una rutina no puede establecer su propia conexión ni restablecer la conexión de la aplicación llamante (SQLCODE -751, SQLSTATE 38003).

Únicamente un procedimiento almacenado definido como MODIFIES SQL DATA puede emitir sentencias COMMIT y ROLLBACK. Los otros tipos de rutinas (UDF y métodos) no pueden emitir COMMIT ni ROLLBACK (SQLCODE -751, SQLSTATE 38003). Aunque un procedimiento almacenado definido como MODIFIES SQL DATA puede intentar confirmar (COMMIT) o retrotraer (ROLLBACK) una transacción, es recomendable que COMMIT o ROLLBACK se realicen desde la aplicación llamante, a fin de que los cambios no se confirmen inesperadamente. Los procedimientos almacenados no podrán emitir sentencias COMMIT ni ROLLBACK si el procedimiento se ha invocado desde una aplicación que ha establecido una conexión de tipo 2 con la base de datos.

Asimismo, sólo los procedimientos almacenados definidos como MODIFIES SQL DATA pueden establecer sus propios puntos de salvaguarda y retrotraer su propio trabajo dentro del punto de salvaguarda. Los otros tipos de rutinas (UDF y métodos) no pueden establecer sus propios puntos de salvaguarda. Un punto de salvaguarda creado dentro de un procedimiento almacenado no se libera cuando finaliza el procedimiento. La aplicación podrá retrotraer el punto de salvaguarda. De forma similar, un procedimiento almacenado podría retrotraer un punto de salvaguarda definido en la aplicación. DB2 liberará implícitamente cualquier punto de salvaguarda establecido por la rutina cuando realice la devolución.

Una rutina puede informar a DB2 sobre si la asignación de un valor de SQLSTATE al argumento sqlstate que DB2 le pasa ha resultado satisfactoria o no. Algunos estilos de parámetros (PARAMETER STYLE JAVA, GENERAL y GENERAL WITH NULLS) no dan soporte al intercambio de valores de SQLSTATE.

Si, al manejar el SQL emitido por una rutina, DB2 encuentra un error, devuelve ese error a la rutina, igual que hace con cualquier aplicación. Para los errores normales del usuario, la rutina tiene la oportunidad de emprender una acción alternativa o correctiva. Por ejemplo, si una rutina intenta realizar inserciones (INSERT) en una tabla y obtiene un error de clave duplicada (SQLCODE -813), en lugar de ello puede decidir actualizar (UPDATE) la fila existente de la tabla.

Sin embargo, se pueden producir determinados errores más graves que hagan imposible que DB2 prosiga de forma normal. Son ejemplos de estos casos los puntos muertos, las anomalías en la partición de la base de datos o las interrupciones del usuario. Algunos de estos errores se propagan hacia arriba, hasta la aplicación llamante. Otros errores graves que están relacionados con la unidad de trabajo recorren todo el camino hasta (a) la aplicación o (b) un procedimiento almacenado que tiene permitido emitir sentencias de control de transacciones (COMMIT o ROLLBACK), el que se encuentre antes en el retroceso.

Si se produce uno de estos errores durante la ejecución del SQL emitido por una rutina, se devuelve el error a la rutina, pero DB2 recuerda que se ha producido un error grave. Además, en este caso, DB2 hará fracasar automáticamente (SQLCODE -20139, SQLSTATE 51038) todo el SQL posterior emitido por esta rutina y por las posibles rutinas llamantes. La única excepción a esto es si el error sólo retrocede hasta el procedimiento almacenado más remoto que tiene permitido emitir sentencias de control de transacciones. En tal caso, este procedimiento almacenado puede continuar emitiendo SQL.

Las rutinas pueden emitir SQL tanto estático como dinámico y, en cualquier caso, se tienen que precompilar y enlazar si se utiliza SQL incorporado. Para el SQL estático, la información utilizada en el proceso de precompilación/enlace es igual que para cualquier aplicación cliente que utilice SQL incorporado. Para el SQL dinámico, se puede utilizar la opción de precompilación/enlace DYNAMICRULES para controlar el esquema actual y el ID de autentificación actual del ID dinámico incorporado. Este comportamiento es distinto para las rutinas y las aplicaciones.

Se respeta el nivel de aislamiento definido para los paquetes de rutinas o sentencias. Esto puede tener como consecuencia que una rutina se ejecute a un nivel de aislamiento más restrictivo o más generoso que la aplicación llamante. Es importante tomar en consideración este aspecto al llamar a una rutina que tenga un nivel de aislamiento menor al de la sentencia que realiza la llamada. Por ejemplo, si se llama a una función de estabilidad del cursor desde una aplicación de lectura repetible, la UDF puede mostrar características de lectura no repetible.

La aplicación o la rutina que realizan la invocación no se ven afectadas por los cambios efectuados por la rutina en valores de registro especiales. La rutina hereda los registros especiales actualizables del invocador. Los cambios en los registros especiales actualizables no se devuelven al invocador. Los registros especiales no actualizables obtendrán su valor por omisión. Si desea más detalles sobre los registros especiales actualizables y los no actualizables, consulte el tema relacionado "Registros especiales".

Las rutinas pueden abrir (OPEN), captar (FETCH) y cerrar (CLOSE) cursores igual que las aplicaciones cliente. Las diversas invocaciones (por ejemplo, en el caso de

una repetición) de la misma función obtienen, cada una de ellas, su propia instancia del cursor. Las UDF y los métodos deben cerrar sus cursores antes de que finalice la sentencia que realiza la invocación; de lo contrario, se producirá un error (SQLCODE -472, SQLSTATE 24517). La llamada final de una UDF o de un método es un buen momento para cerrar los cursores que permanezcan abiertos. Cualquier cursor abierto que no se cierre antes de la finalización en un procedimiento almacenado se devolverá a la aplicación cliente o a la rutina llamante como conjuntos de resultados.

Los argumentos que se pasan a las rutinas no se tratan automáticamente como variables del lenguaje principal. Esto significa que, para que una rutina utilice un parámetro como una variable del lenguaje principal en su SQL, debe declarar su propia variable del lenguaje principal y copiar el valor del parámetro a esta variable.

Nota: Las rutinas que contienen SQL incorporado se tienen que precompilar y enlazar con la opción DATETIME establecida en ISO.

Tareas relacionadas:

- "Personalización de opciones de precompilación y de vinculación para procedimientos SQL" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Información relacionada:

- "Sentencia CREATE FUNCTION" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE PROCEDURE" en la publicación *Consulta de SQL, Volumen 2*
- "Mandato BIND" en la publicación *Consulta de mandatos*
- "Sentencia CREATE FUNCTION (Tabla externa OLE DB)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE FUNCTION (Escalar de SQL, tabla o fila)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE FUNCTION (Escalar externa)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE FUNCTION (Tabla externa)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE FUNCTION (Con origen o plantilla)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencias de SQL que se permiten en rutinas" en la publicación *Consulta de SQL, Volumen 1*
- "Sentencia CREATE PROCEDURE (Externo)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE PROCEDURE (SQL)" en la publicación *Consulta de SQL, Volumen 2*
- "Registros especiales" en la publicación *Consulta de SQL, Volumen 1*

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 2. 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 3. 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.

Tabla 3. Definiciones de comportamientos de sentencias de SQL dinámico (continuación)

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
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 publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente*
- "Autorizaciones y enlace de rutinas que contienen SQL" en la página 38

Rutinas de ejecución en el lenguaje común .NET

Los apartados siguientes describen cómo escribir rutinas .NET que se ejecuten según la ejecución en el lenguaje común .NET Framework.

Rutinas de ejecución en el lenguaje común (CLR)

En DB2®, una rutina de ejecución en el lenguaje común (CLR) es una rutina externa creada ejecutando una sentencia CREATE PROCEDURE o CREATE FUNCTION que hace referencia a un ensamblaje .NET como su cuerpo de código externo.

Los términos siguientes son importantes en el contexto de las rutinas CLR:

.NET Framework
: Un entorno de desarrollo de aplicaciones de Microsoft® que comprende tanto la CLR como la biblioteca de clases de .NET Framework diseñada para proporcionar un entorno de programación coherente con miras al desarrollo e integración de partes de código.

Ejecución en el lenguaje común (CLR)
: El intérprete de ejecución para todas las aplicaciones .NET Framework.

lenguaje intermedio (IL)
: Tipo de código de bytes compilado que se interpreta mediante la CLR de .NET Framework. El código fuente de todos los lenguajes compatibles con .NET se compila en el código de bytes IL.

ensamblaje
: Un archivo que contiene código de bytes IL. Puede ser una biblioteca o un ejecutable.

Es posible implementar rutinas CLR en los lenguajes que se puedan compilar en un ensamblaje IL. Estos lenguajes incluyen los siguientes, pero no están limitados a ellos: Managed C++, C#, Visual Basic y J#.

Antes de desarrollar una rutina CLR, es importante comprender los conceptos básicos de las rutinas y las características exclusivas y específicas de las rutinas CLR. Para informarse más acerca de las rutinas y de las rutinas CLR, consulte:

- "Rutinas en el desarrollo de aplicaciones" en la página 3
- "Tipos de datos de SQL soportados para el Proveedor de datos DB2 .NET" en la página 120
- "Parámetros de las rutinas CLR" en la página 121
- "Devolución de conjuntos de resultados desde procedimientos CLR" en la página 125
- "Restricciones de las rutinas CLR" en la página 126
- "Errores relacionados con rutinas CLR" en la página 128

Desarrollar una rutina CLR es fácil. Para obtener instrucciones paso a paso sobre cómo desarrollar una rutina CLR y ejemplos completos, consulte:

- "Creación de rutinas CLR"
- "Ejemplos de procedimientos CLR en C#" en la página 130
- "Ejemplos de funciones CLR definidas por el usuario en C#" en la página 152

Conceptos relacionados:

- "Rutinas en el desarrollo de aplicaciones" en la página 3
- "Estilos de parámetros para rutinas externas" en la página 95
- "SQL en rutinas externas" en la página 110
- "Tipos de rutinas (procedimientos, funciones, métodos)" en la página 5
- "Autorizaciones y enlace de rutinas que contienen SQL" en la página 38

Tareas relacionadas:

- "Creación de rutinas Common Language Runtime (CLR) .NET" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

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"

Creación de rutinas CLR

Los procedimientos y funciones que hacen referencia a un ensamblaje IL se crean de la misma forma que cualquier rutina externa. Elegirá implementar una rutina externa en un lenguaje .NET si:

- Desea encapsular lógica compleja en una rutina que acceda a la base de datos o que realice una acción fuera de la base de datos.
- Necesita que la lógica encapsulada se invoque desde cualquiera de estos elementos: diversas aplicaciones, el CLP, otra rutina (procedimiento, función (UDF) o método) o un activador.

- Se siente más cómodo al codificar esta lógica en un lenguaje .NET.

Requisitos previos:

- Conocimiento de la implementación de rutinas CLR. Para informarse sobre las rutinas CLR en general y sobre las características CLR, consulte:
  - "Rutinas de ejecución en el lenguaje común (CLR)" en la página 116
- El servidor de bases de datos debe ejecutar un sistema operativo Windows que dé soporte a Microsoft .NET Framework.
- El producto .NET Framework, versión 1.1, debe estar instalado en el servidor. El producto .NET Framework está disponible de forma independiente o como parte de Microsoft .NET Framework 1.1 Software Development Kit.
- Deben estar instaladas las versiones siguientes de DB2:
  - Servidor: DB2 8.2 o un release posterior.
  - Cliente: Cualquier cliente que se pueda conectar a una instancia de DB2 8.2 será capaz de invocar una rutina CLR. Es recomendable instalar DB2 Versión 7.2 o un release posterior en el cliente.
- Autorización para ejecutar la sentencia CREATE correspondiente a la rutina externa. Si desea saber cuáles son los privilegios necesarios para ejecutar la sentencia CREATE PROCEDURE o CREATE FUNCTION, consulte los detalles relativos a la sentencia pertinente.

Restricciones:

Para obtener una lista de las restricciones asociadas con rutinas CLR, consulte:

- "Restricciones de las rutinas CLR" en la página 126

Procedimiento:

1. Codifique la lógica de la rutina en cualquier lenguaje CLR soportado.
   - Para obtener información general sobre las rutinas .NET CLR y sus características, consulte los temas referidos en el apartado Requisitos previos.
   - Utilice o importe `IBM.Data.DB2` si la rutina ha de ejecutar SQL.
   - Declare las variables del lenguaje principal y los parámetros correctamente utilizando tipos de datos que se correlacionen con tipos de datos de SQL de DB2. Para conocer la correlación entre los tipos de datos de DB2 y .NET:
     - "Tipos de datos de SQL soportados para el Proveedor de datos DB2 .NET" en la página 120
   - Los parámetros y los indicadores de nulo de parámetro se deben declarar utilizando uno de los estilos de parámetros soportados por DB2 y de acuerdo con los requisitos de los parámetros de las rutinas .NET CLR. Asimismo, las áreas reutilizables para las UDF y la clase DBINFO pasan a las rutinas CLR como parámetros. Si desea más información sobre los parámetros y las declaraciones de prototipo, consulte:
     - "Parámetros de las rutinas CLR" en la página 121
   - Si la rutina es un procedimiento y desea devolver un conjunto de resultados al llamante de la rutina, no es necesario ningún parámetro para el conjunto de resultados. Si desea más información sobre la devolución de conjuntos de resultados desde rutinas CLR:
     - "Devolución de conjuntos de resultados desde procedimientos CLR" en la página 125
   - Establezca un valor de retorno de rutina si es necesario. Las funciones escalares CLR requieren que se establezca un valor de retorno antes de la

devolución. Las funciones de tabla CLR requieren que se especifique un código de retorno como parámetro de salida para cada invocación de la función de tabla. Los procedimientos CLR no realizan una devolución con un valor de retorno.

2. Cree el código en un ensamblaje de lenguaje intermedio (IL) para que se ejecute mediante la CLR. A fin de obtener información sobre cómo crear rutinas CLR .NET que accedan a DB2, consulte el enlace relacionado:
   - Creación de rutinas de ejecución en el lenguaje común
3. Copie el ensamblaje en el *directorio de función* de DB2 del servidor de bases de datos. Es recomendable almacenar ensamblajes o bibliotecas asociadas con rutinas de DB2 en el directorio de función. Para conocer más acerca del directorio de función, consulte la cláusula EXTERNAL de una de las sentencias siguientes: CREATE PROCEDURE o CREATE FUNCTION.

   Puede copiar el ensamblaje en otro directorio del servidor si lo desea, pero, para invocar satisfactoriamente la rutina, debe anotar el nombre de vía de acceso completamente calificado del ensamblaje porque lo necesitará en el paso siguiente.
4. Ejecute de forma dinámica o estática la sentencia CREATE de lenguaje SQL correspondiente para el tipo de rutina: CREATE PROCEDURE o CREATE FUNCTION.
   - Especifique la cláusula `LANGUAGE` con el valor: `CLR`.
   - Especifique la cláusula `PARAMETER STYLE` con el nombre del estilo de parámetros soportado que se ha implementado en el código de la rutina.
   - Especifique la cláusula `EXTERNAL` con el nombre del ensamblaje que se ha de asociar con la rutina utilizando uno de los valores siguientes:
     – el nombre de vía de acceso completamente calificado del ensamblaje de rutina.
     – el nombre de vía de acceso relativo del ensamblaje de rutina en relación con el directorio de función.

     Por omisión, DB2 buscará el ensamblaje por el nombre en el directorio de función, a menos que se especifique un nombre de vía de acceso completamente calificado o relativo para la biblioteca en la cláusula EXTERNAL.

     Cuando se ejecute la sentencia CREATE, si DB2 no encuentra el ensamblaje especificado en la cláusula EXTERNAL, se recibirá un error (SQLCODE -20282) con el código de razón 1.
   - Especifique DYNAMIC RESULT SETS con el valor 1 si la rutina es un procedimiento y ha de devolver un conjunto de resultados al llamante.
   - No es posible especificar la cláusula NOT FENCED para los procedimientos CLR. Por omisión, los procedimientos CLR se ejecutan como procedimientos FENCED.

Para invocar la rutina CLR, consulte "Invocación de la rutina" en la página 209.

Conceptos relacionados:

- "Rutinas de ejecución en el lenguaje común (CLR)" en la página 116
- "Parámetros de las rutinas CLR" en la página 121
- "Invocación de la rutina" en la página 209
- "Estilos de parámetros para rutinas externas" en la página 95
- "Áreas reutilizables para UDF y métodos" en la página 58
- "SQL en rutinas externas" en la página 110

- "Tipos de rutinas (procedimientos, funciones, métodos)" en la página 5
- "Conjuntos de resultados de procedimiento" en la página 47

Tareas relacionadas:

- "Devolución de conjuntos de resultados desde procedimientos CLR" en la página 125
- "Creación de rutinas Common Language Runtime (CLR) .NET" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de rutinas en la base de datos" en la página 35
- "Depuración de rutinas" en la página 42

Información relacionada:

- "Restricciones de las rutinas CLR" en la página 126
- "Tipos de datos de SQL soportados para el Proveedor de datos DB2 .NET" en la página 120

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"

Tipos de datos de SQL soportados para el Proveedor de datos DB2 .NET

La tabla siguiente lista las correlaciones entre los tipos de datos `DB2Type` en el Proveedor de datos DB2 .NET, el tipo de datos DB2 y el tipo de datos .NET Framework correspondiente:

Tabla 4. Correlación de tipos de datos DB2 con tipos de datos .NET

DB2Type Enum	Tipo de datos DB2	Tipo de datos .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[]

Tabla 4. Correlación de tipos de datos DB2 con tipos de datos .NET (continuación)

DB2Type Enum	Tipo de datos DB2	Tipo de datos .NET
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 de ejecución en el lenguaje común DB2 .NET. Sólo están soportados en las aplicaciones cliente.

Nota: La estructura `dbinfo` pasa a las funciones y procedimientos CLR como un parámetro. El área reutilizable y el tipo de llamada para las UDF CLR también pasan a las rutinas CLR como parámetros. Si desea información sobre los tipos de datos CLR adecuados para estos parámetros, consulte el tema relacionado:

- Parámetros de las rutinas CLR

Conceptos relacionados:

- "Estilos de parámetros para rutinas externas" en la página 95
- "Rutinas de ejecución en el lenguaje común (CLR)" en la página 116
- "Parámetros de las rutinas CLR" en la página 121

Tareas relacionadas:

- "Pase de parámetros de tipo estructurado a rutinas externas" en la página 314
- "Creación de rutinas CLR" en la página 117
- "Ejemplos de funciones CLR definidas por el usuario en C#" en la página 152
- "Ejemplos de procedimientos CLR en C#" en la página 130

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"

Parámetros de las rutinas CLR

La declaración de parámetros de las rutinas CLR debe cumplir con los requisitos de uno de los estilos de parámetros soportados y debe respetar los requisitos de palabra clave de parámetro del lenguaje .NET determinado que se utilice para la rutina. Si la rutina ha de emplear un área reutilizable o la estructura `dbinfo` o ha

de tener la interfaz de parámetros PROGRAM TYPE MAIN, existen detalles adicionales a considerar. Este tema aborda todas las consideraciones sobre los parámetros CLR.

Estilos de parámetros soportados para las rutinas CLR:

El estilo de parámetros de la rutina se debe especificar durante la creación de la rutina en la cláusula EXTERNAL de la sentencia CREATE para la rutina. El estilo de parámetros se debe reflejar adecuadamente en la implementación del código de la rutina CLR externa. Están soportados los estilos de parámetros siguientes de DB2® para las rutinas CLR:

- `SQL` (Soportado para procedimientos y funciones)
- `GENERAL` (Soportado para procedimientos únicamente)
- `GENERAL WITH NULLS` (Soportado para procedimientos únicamente)
- `DB2SQL` (Soportado para procedimientos y funciones)

Si desea más información sobre estos estilos de parámetros, consulte:

- "Estilos de parámetros para rutinas externas" en la página 95

Indicadores de nulo de los parámetros de rutinas CLR:

Si el estilo de parámetros elegido para una rutina CLR requiere que se especifiquen indicadores de nulo para los parámetros, los indicadores de nulo se han de pasar a la rutina CLR como valores de tipo System.Int16 o en un valor System.Int16[] cuando el estilo de parámetros exija un vector de indicadores de nulo.

Cuando el estilo de parámetros impone que se pasen los indicadores de nulo a la rutina como parámetros diferenciados, tal como requiere el estilo de parámetros SQL, se necesita un indicador de nulo System.Int16 para cada parámetro.

En los lenguajes .NET, los parámetros diferenciados deben ir precedidos de una palabra clave para indicar si el parámetro se pasa por valor o por referencia. La misma palabra clave que se utilice para un parámetro de rutina se debe utilizar para el parámetro de indicador de nulo asociado. Las palabras clave que indican si un argumento se pasa por valor o por referencia se describen con más detalle a continuación.

Si desea más información sobre el estilo de parámetros SQL y los otros estilos de parámetros soportados, consulte:

- "Estilos de parámetros para rutinas externas" en la página 95

Pase de parámetros de rutinas CLR por valor o por referencia:

Las rutinas de lenguaje .NET que se compilan en el código de bytes del lenguaje intermedio (IL) requieren que los parámetros vayan precedidos de palabras clave que indiquen las propiedades determinadas del parámetro, tales como si el parámetro se pasa por valor o por referencia, o si es un parámetro de sólo entrada o de sólo salida.

Las palabras clave de parámetro son específicas del lenguaje .NET. Por ejemplo, para pasar un parámetro por referencia en `C#`, la palabra clave de parámetro es `ref`, mientras que, en `Visual Basic`, un parámetro por referencia se indica

mediante la palabra clave byRef. Las palabras clave se deben emplear para indicar el uso del parámetro de SQL (IN, OUT, INOUT) que se ha especificado en la sentencia CREATE para la rutina.

Se imponen las normas siguientes al aplicar palabras clave de parámetro a los parámetros de rutinas de lenguaje .NET en las rutinas de DB2:

- Los parámetros de tipo IN se deben declarar *sin* palabra clave de parámetro en C#, y se deben declarar con la palabra clave byVal en Visual Basic.
- Los parámetros de tipo INOUT se deben declarar con la palabra clave específica del lenguaje que indique que el parámetro se pasa por referencia. En C#, la palabra clave adecuada es ref. En Visual Basic, la palabra clave adecuada es byRef.
- Los parámetros de tipo OUT se deben declarar con la palabra clave específica del lenguaje que indique que el parámetro es de sólo salida. En C#, utilice la palabra clave out. En Visual Basic, el parámetro se debe declarar con la palabra clave byRef. Los parámetros de sólo salida siempre deben tener asignado un valor antes de que la rutina vuelva al llamante. Si la rutina no asigna un valor al parámetro de sólo salida, se generará un error cuando se compile la rutina .NET.

A continuación, se muestra el aspecto de un prototipo de procedimiento en C# del estilo de parámetros SQL para una rutina que devuelve un solo parámetro de salida language.

```
public static void Counter  (out String language,
                             out Int16  languageNullInd,
                             ref String sqlState,
                                 String funcName,
                                 String funcSpecName,
                             ref String sqlMsgString,
                             ref Byte[] scratchPad,
                                 Int32  callType);
```

Es evidente que se implementa el estilo de parámetros SQL por el parámetro de indicador de nulo adicional, languageNullInd, asociado con el parámetro de salida language, los parámetros para pasar el SQLSTATE, el nombre de rutina, el nombre de rutina específico y un mensaje de error de SQL opcional definido por el usuario. Se han especificado palabras clave para los parámetros del modo siguiente:

- En C#, no es necesaria ninguna palabra clave de parámetro para los parámetros de sólo entrada.
- En C#, la palabra clave 'out' indica que la variable es un parámetro de sólo salida y que el llamante no ha inicializado su valor.
- En C#, la palabra clave 'ref' indica que el llamante ha inicializado el parámetro y que la rutina puede modificar este valor opcionalmente.

Consulte la documentación específica del lenguaje .NET en relación con el pase de parámetros para informarse sobre las palabras clave de parámetro en ese lenguaje.

Nota:

> DB2 controla la asignación de memoria para todos los parámetros y mantiene las referencias CLR a todos los parámetros pasados a una rutina o salidos de ella.

No es necesario ningún marcador de parámetro para los conjuntos de resultados de procedimiento:

Los marcadores de parámetros no son necesarios en la declaración de un procedimiento para un conjunto de resultados que se va a devolver al llamante. Cualquier sentencia de cursor que no se haya cerrado desde dentro de un procedimiento almacenado CLR se devolverá a su llamante como conjunto de resultados.

Si desea más información sobre los conjuntos de resultados en las rutinas CLR, consulte:

- "Devolución de conjuntos de resultados desde procedimientos CLR" en la página 125

Estructura Dbinfo como parámetro CLR:

La estructura `dbinfo` utilizada para pasar parámetros adicionales de información de base de datos a y desde una rutina está soportada para las rutinas CLR mediante el uso de una clase `dbinfo` de IL. Esta clase contiene todos los elementos que se encuentran en la estructura `sqludf_dbinfo` del lenguaje `C`, a excepción de los campos de longitud asociados con las series. La longitud de cada serie se puede encontrar utilizando la propiedad de lenguaje .NET `Length` de la serie determinada.

Para acceder a la clase `dbinfo`, simplemente incluya el ensamblaje `IBM®.Data.DB2` en el archivo que contenga la rutina y añada un parámetro del tipo `sqludf_dbinfo` a la signatura de la rutina, en la posición especificada por el estilo de parámetros utilizado.

Área reutilizable de UDF como parámetro CLR:

Si se solicita un área reutilizable para una función definida por el usuario, se pasa a la rutina como parámetro `System.Byte[]` del tamaño especificado.

Parámetro de llamada final o de tipo de llamada de UDF CLR:

Para las funciones definidas por el usuario que han solicitado un parámetro de llamada final o para las funciones de tabla, el parámetro de tipo de llamada se pasa a la rutina como tipo de datos `System.Int32`.

PROGRAM TYPE MAIN está soportado para los procedimientos CLR:

PROGRAM TYPE MAIN está soportado para los procedimientos .NET CLR. Los procedimientos definidos para el uso de Program Type MAIN deben tener la signatura siguiente:

```
void functionname(Int32 NumParams, Object[] Params)
```

Conceptos relacionados:

- "Estilos de parámetros para rutinas externas" en la página 95
- "Modalidades de parámetros de procedimiento" en la página 46
- "Áreas reutilizables para UDF y métodos" en la página 58
- "Conjuntos de resultados de procedimiento" en la página 47
- "Manejo de los parámetros en los procedimientos de PROGRAM TYPE MAIN o PROGRAM TYPE SUB" en la página 56

Tareas relacionadas:

- "Pase de parámetros de tipo estructurado a rutinas externas" en la página 314
- "Ejemplos de funciones CLR definidas por el usuario en C#" en la página 152
- "Ejemplos de procedimientos CLR en C#" en la página 130
- "Devolución de conjuntos de resultados desde procedimientos CLR" en la página 125

Información relacionada:

- "Tipos de datos de SQL soportados para el Proveedor de datos DB2 .NET" en la página 120

Devolución de conjuntos de resultados desde procedimientos CLR

Es posible desarrollar procedimientos CLR que devuelvan conjuntos de resultados a la rutina o aplicación que realiza la llamada. No se pueden devolver conjuntos de resultados desde las funciones CLR (UDF).

La representación en .NET de un conjunto de resultados es un objeto `DB2DataReader` que se puede devolver desde una de las diversas llamadas de ejecución de un objeto `DB2Command`. Se puede devolver cualquier objeto `DB2DataReader` cuyo método `Close()` no se haya llamado explícitamente antes de la devolución del procedimiento. El orden en que se devuelven los conjuntos de resultados al llamante es el mismo orden en que se han creado las instancias de los objetos `DB2DataReader`. No se requieren parámetros adicionales en la definición de función para devolver un conjunto de resultados.

Requisitos previos:

Un conocimiento general de cómo se crean las rutinas CLR le ayudará a seguir los pasos del procedimiento siguiente sobre la devolución de resultados de un procedimiento CLR.

"Creación de rutinas CLR" en la página 117

Procedimiento:

Para devolver un conjunto de resultados de un procedimiento CLR:

1. En la sentencia CREATE PROCEDURE para la rutina CLR, debe especificar, junto con cualquier otra cláusula apropiada, la cláusula DYNAMIC RESULT SETS con un valor que equivalga al número de conjuntos de resultados que debe devolver el procedimiento.
2. Los marcadores de parámetros no son necesarios en la declaración de un procedimiento para un conjunto de resultados que se va a devolver al llamante.
3. En la implementación en el lenguaje .NET de la rutina CLR, cree un objeto `DB2Connection`, un objeto `DB2Command` y un objeto `DB2Transaction`. El objeto `DB2Transaction` es responsable de la retrotracción y confirmación de las transacciones de base de datos.
4. Inicialice la propiedad Transaction del objeto `DB2Command` para el objeto `DB2Transaction`.
5. Asigne una consulta de serie a la propiedad `CommandText` del objeto `DB2Command` que defina el conjunto de resultados que desea devolver.

6. Cree una instancia de un `DB2DataReader` y asígnele el resultado de la invocación del método `ExecuteReader` del objeto `DB2Command`. El conjunto de resultados de la consulta estará incluido en el objeto `DB2DataReader`.
7. No ejecute el método `Close()` del objeto `DB2DataReader` en ningún punto anterior a la devolución del procedimiento al llamante. El objeto `DB2DataReader` se devolverá todavía abierto como un conjunto de resultados al llamante.

   Cuando se deja abierto más de un `DB2DataReader` después de la devolución de un procedimiento, los `DB2DataReader` se devuelven al llamante siguiendo el orden de su creación. Sólo se devolverá al llamante el número de conjuntos de resultados especificado en la sentencia CREATE PROCEDURE.
8. Compile el procedimiento de lenguaje .NET CLR e instale el ensamblaje en la ubicación especificada por la cláusula EXTERNAL de la sentencia CREATE PROCEDURE. Ejecute la sentencia CREATE PROCEDURE para el procedimiento CLR, si todavía no lo ha hecho.
9. Una vez que haya instalado el ensamblaje del procedimiento CLR en la ubicación adecuada y haya ejecutado la sentencia CREATE PROCEDURE satisfactoriamente, puede invocar el procedimiento con la sentencia CALL para ver la devolución de conjuntos de resultados al llamante.

   Si desea obtener información sobre la llamada a procedimientos y otros tipos de rutinas:

   - "Invocación de la rutina" en la página 209

Nota: Especificación de DYNAMIC RESULT SETS con un valor superior a 1. Sólo se puede devolver un conjunto de resultados dinámico desde los procedimientos CLR en este momento. Para obtener detalles sobre esta restricción, consulte:

- "Restricciones de las rutinas CLR"

Conceptos relacionados:

- "Invocación de la rutina" en la página 209
- "Modalidades de parámetros de procedimiento" en la página 46
- "Conjuntos de resultados de procedimiento" en la página 47
- "Rutinas de ejecución en el lenguaje común (CLR)" en la página 116

Tareas relacionadas:

- "Creación de rutinas CLR" en la página 117

Información relacionada:

- "Restricciones de las rutinas CLR" en la página 126

Restricciones de las rutinas CLR

Las restricciones generales de implementación que se aplican a todas las rutinas externas o a determinadas clases de rutinas (procedimiento o UDF) también se aplican a las rutinas CLR. Existen algunas restricciones que son particulares de las rutinas CLR. Estas restricciones se listan aquí.

La sentencia CREATE METHOD con la cláusula LANGUAGE CLR no está soportada:

No se pueden crear métodos externos para tipos estructurados de DB2 que hacen referencia a un ensamblaje CLR. El uso de una sentencia CREATE METHOD que especifique la cláusula LANGUAGE con el valor CLR no está soportado.

Los procedimientos CLR no se pueden implementar como procedimientos NOT FENCED:

Los procedimientos CLR no se pueden ejecutar como procedimientos no protegidos. La sentencia CREATE PROCEDURE para un procedimiento CLR no puede especificar la cláusula NOT FENCED.

En este momento, los procedimientos CLR pueden devolver un máximo de un conjunto de resultados:

El número máximo de conjuntos de resultados que puede devolver un procedimiento CLR está limitado al número máximo de objetos `DB2DataReader` que el proveedor de datos (`IBM.Data.DB2`) puede permitir tener abiertos simultáneamente dentro de una conexión. El número máximo de los que puede haber abiertos en este momento es de uno. Por ello, sólo se puede devolver un conjunto de resultados de un procedimiento CLR.

Si una sentencia CREATE PROCEDURE de un procedimiento CLR especifica la cláusula DYNAMIC RESULT SETS con un valor superior a la unidad, no se emitirá ningún error cuando se ejecute esta sentencia.

No obstante, durante la ejecución, sólo está permitido un `DB2DataReader` abierto en el procedimiento cuando éste realiza la devolución. Por lo tanto, sólo se devolverá al llamante el único conjunto de resultados asociado con el `DB2DataReader` abierto cuando el procedimiento realice la devolución.

La precisión decimal máxima es de 29 y la escala decimal máxima es de 28 en una rutina CLR:

El tipo de datos decimal en DB2 se representa con una precisión de 31 dígitos y una escala de 28 dígitos. En los lenguajes de programación .NET, el tipo de datos decimal se representa con una precisión de 29 dígitos y una escala de 28 dígitos. Por consiguiente, para evitar el truncamiento de datos, las rutinas CLR externas de DB2 no deben especificar un valor decimal que sobrepase una precisión de 29 dígitos y una escala de 28 dígitos.

Si es necesario que su rutina manipule valores decimales con la precisión y escala máximas soportadas por DB2, puede implementar la rutina externa en un lenguaje de programación distinto, como, por ejemplo, C o Java.

Tipos de datos no soportados en las rutinas CLR:

Los siguientes tipos de datos de SQL de DB2 no están soportados en las rutinas CLR:

- LONG VARCHAR
- LONG VARCHAR FOR BIT DATA
- LONG GRAPHIC
- DATALINK
- ROWID

Ejecución de una rutina CLR de 32 bits en una instancia de 64 bits:

Las rutinas CLR no se pueden ejecutar en instancias de 64 bits, porque .NET Framework no se puede instalar en sistemas operativos de 64 bits en este momento.

Conceptos relacionados:

- "Rutinas de ejecución en el lenguaje común (CLR)" en la página 116
- "Parámetros de las rutinas CLR" en la página 121

Tareas relacionadas:

- "Creación de rutinas CLR" en la página 117
- "Devolución de conjuntos de resultados desde procedimientos CLR" en la página 125

Errores relacionados con rutinas CLR

Todas las rutinas externas comparten una implementación común en general, y se pueden producir algunos errores de DB2 específicos de las rutinas CLR. Esta consulta lista los errores más probables que se pueden encontrar, por el SQLCODE o comportamiento, con algunas sugerencias para la depuración. Los errores de DB2 relacionados con rutinas se pueden clasificar del modo siguiente:

Errores de creación de la rutina
Errores que se producen cuando se ejecuta la sentencia CREATE para la rutina.

Errores de ejecución de la rutina
Errores que se producen durante la invocación o ejecución de la rutina.

Independientemente de cuándo DB2 emite un error relacionado con una rutina de DB2, el texto del mensaje de error detalla la causa del error y la acción que el usuario debe emprender para resolver el problema. Hallará información adicional sobre escenarios de los errores de rutinas en el archivo de registro de diagnósticos `db2diag.log`.

Errores de creación de rutinas CLR:

SQLCODE -451, SQLSTATE 42815
Este error se emite tras un intento de ejecutar una sentencia CREATE TYPE que incluye una declaración de método externo especificando la cláusula LANGUAGE con el valor CLR. No se pueden crear métodos externos de DB2 para tipos estructurados que hagan referencia a un ensamblaje de CLR en este momento. Cambie la cláusula LANGUAGE de forma que especifique un lenguaje soportado para el método e implemente el método en ese lenguaje alternativo.

SQLCODE -449, SQLSTATE 42878
La sentencia CREATE para la rutina CLR contiene una identificación de función o biblioteca de formato no válido en la cláusula EXTERNAL NAME. Para el lenguaje CLR, el valor de la cláusula EXTERNAL debe tomar el formato específico `'<a>:<b>!<c>'`, del modo siguiente:

- `<a>` es el archivo de ensamblaje de CLR en el que está ubicada la clase.
- `<b>` es la clase en la que reside el método a invocar.
- `<c>` es el método a invocar.

No se permiten caracteres en blanco iniciales ni de cola entre las comillas simples, identificadores de objeto y caracteres de separación (por ejemplo,

' <a> ! <b> ' no es válido). Sin embargo, los nombres de vía de acceso y archivo pueden contener espacios en blanco si la plataforma lo permite. Para todos los nombres de archivo, el archivo se puede especificar utilizando la forma abreviada del nombre (ejemplo: `math.dll`) o el nombre de vía de acceso completamente calificado (ejemplo: `d:\udfs\math.dll`). Si se utiliza la forma abreviada del nombre de archivo, si la plataforma es UNIX o si la rutina es LANGUAGE CLR, el archivo debe residir en el directorio de función. Si la plataforma es Windows y la rutina no es una rutina LANGUAGE CLR, el archivo debe residir en la vía de acceso (PATH) del sistema. Las extensiones de archivo (ejemplos: `.a` (en UNIX), `.dll` (en Windows)) siempre se deben incluir en el nombre de archivo.

Errores de ejecución de rutinas CLR:

SQLCODE -20282, SQLSTATE 42724, código de razón 1
No se ha encontrado el ensamblaje externo especificado por la cláusula EXTERNAL en la sentencia CREATE para la rutina.

- Compruebe si la cláusula EXTERNAL especifica el nombre correcto de ensamblaje de la rutina y si el ensamblaje se encuentra en la ubicación especificada. Si la cláusula EXTERNAL no especifica un nombre de vía de acceso completamente calificado para el ensamblaje deseado, DB2 supone que el nombre de vía de acceso que se proporciona es un nombre de vía de acceso relativo para el ensamblaje, en relación con el directorio de función de DB2.

SQLCODE -20282, SQLSTATE 42724, código de razón 2
Se ha encontrado un ensamblaje en la ubicación especificada por la cláusula EXTERNAL de la sentencia CREATE para la rutina, pero, en el ensamblaje, no se ha encontrado ninguna clase que coincida con la clase especificada en la cláusula EXTERNAL.

- Compruebe si el nombre de ensamblaje especificado en la cláusula EXTERNAL es el ensamblaje correcto para la rutina y si existe en la ubicación especificada.
- Compruebe si el nombre de clase especificado en la cláusula EXTERNAL es el nombre de clase correcto y si existe en el ensamblaje especificado.

SQLCODE -20282, SQLSTATE 42724, código de razón 3
Se ha encontrado un ensamblaje en la ubicación especificada por la cláusula EXTERNAL de la sentencia CREATE para la rutina, el cual tenía una definición de clase que coincidía correctamente, pero la signatura del método de la rutina no coincide con la signatura de la rutina especificada en la sentencia CREATE para la rutina.

- Compruebe si el nombre de ensamblaje especificado en la cláusula EXTERNAL es el ensamblaje correcto para la rutina y si existe en la ubicación especificada.
- Compruebe si el nombre de clase especificado en la cláusula EXTERNAL es el nombre de clase correcto y si existe en el ensamblaje especificado.
- Compruebe si la implementación del estilo de parámetros coincide con el estilo de parámetros especificado en la sentencia CREATE para la rutina.
- Compruebe si el orden de la implementación de parámetros coincide con el orden de la declaración de parámetros de la sentencia CREATE para la rutina y si se respetan los requisitos adicionales de parámetros del estilo de parámetros.

- Compruebe si los tipos de datos de los parámetros de SQL están correlacionados correctamente con los tipos de datos soportados en CLR .NET.

SQLCODE -4301, SQLSTATE 58004, código de razón 5 ó 6

Se ha producido un error al intentar el inicio o la comunicación de un intérprete de .NET. DB2 no ha podido cargar una biblioteca .NET dependiente [código de razón 5] o ha fallado una llamada al intérprete de .NET [código de razón 6].

- Asegúrese de que la instancia de DB2 está configurada correctamente para ejecutar una función o procedimiento .NET (`mscoree.dll` debe estar presente en la vía de acceso (`PATH`) del sistema). Asegúrese de que `db2clr.dll` está presente en el directorio `sqllib/bin` y de que `IBM.Data.DB2` está instalado en la antememoria del ensamblaje global. Si no están, asegúrese de que `.NET Framework` versión 1.1, o una versión posterior, se haya instalado en el servidor de bases de datos y de que dicho servidor esté ejecutando DB2 versión 8.2 o un release posterior.

SQLCODE -4302, SQLSTATE 38501

Se ha producido una excepción no controlada durante la ejecución, en la preparación de la ejecución o después de ejecutar la rutina. Puede ser resultado de un error de programación de la lógica de una rutina no controlado o puede ser resultado de un error de proceso interno.

Conceptos relacionados:

- "SQL en rutinas externas" en la página 110
- "Autorizaciones y enlace de rutinas que contienen SQL" en la página 38
- "Consideraciones sobre seguridad para las rutinas" en la página 27
- "Consideraciones sobre la gestión de bibliotecas y clases" en la página 30
- "Rutinas de ejecución en el lenguaje común (CLR)" en la página 116

Tareas relacionadas:

- "Creación de rutinas CLR" en la página 117

Ejemplos de procedimientos CLR en C#

Una vez comprendidos los conceptos básicos de los procedimientos, también denominados procedimientos almacenados, y los fundamentos de las rutinas de ejecución en el lenguaje común .NET, puede empezar a utilizar procedimientos CLR en sus aplicaciones.

Este tema contiene ejemplos de procedimientos CLR implementados en C# que ilustran los estilos de parámetros soportados, el pase de parámetros, incluida la estructura `dbinfo`, cómo devolver un conjunto de resultados y más información. Para obtener ejemplos de UDF CLR en C#:

- "Ejemplos de funciones CLR definidas por el usuario en C#" en la página 152

Requisitos previos:

Antes de trabajar con los ejemplos de procedimientos CLR, puede ser conveniente que lea los temas sobre los conceptos siguientes:

- "Rutinas de ejecución en el lenguaje común (CLR)" en la página 116
- "Creación de rutinas CLR" en la página 117
- "Rutinas en el desarrollo de aplicaciones" en la página 3

- Creación de rutinas .NET de ejecución en el lenguaje común (CLR)

Los ejemplos siguientes utilizan una tabla denominada `EMPLOYEE` que está incluida en la base de datos `SAMPLE`.

Procedimiento:

Utilice los ejemplos siguientes como referencias al crear sus propios procedimientos CLR en C#:

- "Archivo de código externo C#"
- "Ejemplo 1: Procedimiento en C# del estilo de parámetros GENERAL" en la página 132
- "Ejemplo 2: Procedimiento en C# del estilo de parámetros GENERAL WITH NULLS" en la página 133
- "Ejemplo 3: Procedimiento en C# del estilo de parámetros SQL" en la página 135
- "Ejemplo 4: Procedimiento en C# que devuelve un conjunto de resultados" en la página 138
- "Ejemplo 5: Procedimiento en C# que accede a la estructura dbinfo" en la página 138
- "Ejemplo 6: Procedimiento en C# del estilo PROGRAM TYPE MAIN" en la página 139

Archivo de código externo C#:

Los ejemplos muestran una variedad de implementaciones de procedimientos en C#. Cada ejemplo se compone de dos partes: la sentencia CREATE PROCEDURE y la implementación en código C# externo del procedimiento desde el cual se puede crear el ensamblaje asociado.

El archivo fuente en C# que contiene las implementaciones de procedimientos de los ejemplos siguientes se denomina `gwenProc.cs` y tiene el formato siguiente:

Tabla 5. Formato del archivo de código externo C#

```
using System;
using System.IO;
using IBM.Data.DB2;

namespace bizLogic
{
   class empOps
   {          ...
     // Procedimientos en C#
              ...
   }
}
```

Las inclusiones del archivo se indican al principio del mismo. La inclusión `IBM.Data.DB2` es necesaria si alguno de los procedimientos del archivo contiene SQL. Existe una declaración de espacio de nombres en este archivo y una clase `empOps` que contiene los procedimientos. El uso de espacios de nombres es opcional. Si se utiliza un espacio de nombres, éste debe aparecer en el nombre de vía de acceso de ensamblaje proporcionado en la cláusula EXTERNAL de la sentencia CREATE PROCEDURE.

Es importante tener en cuenta el nombre del archivo, el espacio de nombres y el nombre de la clase, que contiene una implementación de procedimiento determinada. Estos nombres son importantes, ya que la cláusula EXTERNAL de la sentencia CREATE PROCEDURE correspondiente a cada procedimiento debe especificar esta información a fin de que DB2 pueda localizar el ensamblaje y la clase del procedimiento CLR.

Ejemplo 1: Procedimiento en C# del estilo de parámetros GENERAL:

Este ejemplo muestra lo siguiente:

- Sentencia CREATE PROCEDURE para un procedimiento del estilo de parámetros GENERAL
- Código C# para un procedimiento del estilo de parámetros GENERAL

Este procedimiento toma un ID de empleado y una cantidad de bonificación actual como entrada. Recupera el nombre y el salario del empleado. Si la cantidad de bonificación actual es cero, se calcula una nueva bonificación basada en el salario del empleado y se devuelve junto con el nombre y apellidos del empleado. Si no se encuentra el empleado, se devuelve una serie vacía.

Tabla 6. Código para crear un procedimiento en C# del estilo de parámetros GENERAL

```
CREATE PROCEDURE setEmpBonusGEN(IN empID CHAR(6), INOUT bonus Decimal(9,2),
                                OUT empName VARCHAR(60))
SPECIFIC SetEmpBonusGEN
LANGUAGE CLR
PARAMETER STYLE GENERAL
DYNAMIC RESULT SETS 0
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!SetEmpBonusGEN' ;
```

Tabla 6. Código para crear un procedimiento en C# del estilo de parámetros GENERAL (continuación)

```
public static void SetEmpBonusGEN(    String empID,
                                  ref Decimal bonus,
                                  out String empName)
{
   // Declarar las variables locales
   Decimal salary = 0;

   DB2Command myCommand = DB2Context.GetCommand();
   myCommand.CommandText =
                    "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY "
                  + "FROM EMPLOYEE "
                  + "WHERE EMPNO = '" + empID + "';

   DB2DataReader reader = myCommand.ExecuteReader();

   if (reader.Read())  // Si se encuentra el registro de empleado
   {
      // Obtener nombre y apellidos y salario del empleado
      empName = reader.GetString(0) + " " +
                reader.GetString(1) + ". " +
                reader.GetString(2);

      salary = reader.GetDecimal(3);

      if (bonus == 0)
      {
         if (salary > 75000)
         {
            bonus = salary * (Decimal)0.025;
         }
 else
         {
            bonus = salary * (Decimal)0.05;
         }
      }
   }
   else  // No se encuentra el empleado
   {
      empName = "";  // Establecer el parámetro de salida
   }

   reader.Close();
}
```

Ejemplo 2: Procedimiento en C# del estilo de parámetros GENERAL WITH NULLS:
Este ejemplo muestra lo siguiente:

- Sentencia CREATE PROCEDURE para un procedimiento del estilo de parámetros GENERAL WITH NULLS
- Código C# para un procedimiento del estilo de parámetros GENERAL WITH NULLS

Este procedimiento toma un ID de empleado y una cantidad de bonificación actual como entrada. Si el parámetro de entrada no es nulo, recupera el nombre y salario del empleado. Si la cantidad de bonificación actual es cero, se calcula una nueva bonificación basada en el salario y se devuelve junto con el nombre y apellidos del empleado. Si no se encuentran los datos de empleado, se devuelven un entero y una serie NULL.

Tabla 7. Código para crear un procedimiento en C# del estilo de parámetros GENERAL WITH NULLS

```
CREATE PROCEDURE SetEmpbonusGENNULL(IN empID CHAR(6),
                                    INOUT bonus Decimal(9,2),
                                    OUT empName VARCHAR(60))
SPECIFIC SetEmpbonusGENNULL
LANGUAGE CLR
PARAMETER STYLE GENERAL WITH NULLS
DYNAMIC RESULT SETS 0
FENCED
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!SetEmpBonusGENNULL'
;
```

Tabla 7. Código para crear un procedimiento en C# del estilo de parámetros GENERAL WITH NULLS (continuación)

```
public static void SetEmpBonusGENNULL(    String empID,
                                      ref Decimal bonus,
                                      out String empName,
                                          Int16[] NullInds)
{
   Decimal salary = 0;

   if (NullInds[0] == -1) // Comprobar si la entrada es nula
   {
     NullInds[1] = -1;    // Devolver un valor de bonificación NULL
     empName = "";        // Establecer el valor de salida
     NullInds[2] = -1;    // Devolver un valor empName NULL
   }
 else
   {
      DB2Command myCommand = DB2Context.GetCommand();
     myCommand.CommandText =
                      "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY "
                    + "FROM EMPLOYEE "
                    + "WHERE EMPNO = '" + empID + "'";
      DB2DataReader reader = myCommand.ExecuteReader();

      if (reader.Read())  // Si se encuentra el registro de empleado
      {
         // Obtener nombre y apellidos y salario del empleado
         empName = reader.GetString(0) + " "
         +
                   reader.GetString(1) + ". " +
                   reader.GetString(2);
         salary = reader.GetDecimal(3);

         if (bonus == 0)
         {
           if (salary > 75000)
           {
              bonus = salary * (Decimal)0.025;
              NullInds[1] = 0; // Devolver un valor distinto de NULL
           }
 else
           {
              bonus = salary * (Decimal)0.05;
              NullInds[1] = 0; // Devolver un valor distinto de NULL
           }
         }
      }
      else  // No se encuentra el empleado
      {
         empName = "*sdq;;         // Establecer el parámetro de salida
         NullInds[2] = -1;     // Devolver un valor NULL
      }

      reader.Close();
   }
}
```

Ejemplo 3: Procedimiento en C# del estilo de parámetros SQL:
Este ejemplo muestra lo siguiente:

- Sentencia CREATE PROCEDURE para un procedimiento del estilo de parámetros SQL
- Código C# para un procedimiento del estilo de parámetros SQL

Este procedimiento toma un ID de empleado y una cantidad de bonificación actual como entrada. Recupera el nombre y el salario del empleado. Si la cantidad de bonificación actual es cero, se calcula una nueva bonificación basada en el salario y se devuelve junto con el nombre y apellidos del empleado. Si no se encuentra el empleado, se devuelve una serie vacía.

Tabla 8. Código para crear un procedimiento en C# del estilo de parámetros SQL con parámetros

```
CREATE PROCEDURE SetEmpbonusSQL(IN empID CHAR(6),
                                INOUT bonus Decimal(9,2),
                                OUT empName VARCHAR(60))
SPECIFIC SetEmpbonusSQL
LANGUAGE CLR
PARAMETER STYLE SQL
DYNAMIC RESULT SETS 0
FENCED
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!SetEmpBonusSQL' ;
```

Tabla 8. Código para crear un procedimiento en C# del estilo de parámetros SQL con parámetros (continuación)

```
public static void SetEmpBonusSQL(    String empID,
                                   ref Decimal bonus,
                                   out String empName,
                                       Int16  empIDNullInd,
                                   ref Int16  bonusNullInd,
                                   out Int16  empNameNullInd,
                                   ref string sqlStateate,
                                       string funcName,
                                       string specName,
                                   ref string sqlMessageText)
{
   // Declarar las variables locales del lenguaje principal
   Decimal salary eq; 0;

   if (empIDNullInd == -1) // Comprobar si la entrada es nula
   {
      bonusNullInd = -1;   // Devolver un valor de bonificación NULL
      empName = "";
      empNameNullInd = -1; // Devolver un valor empName NULL
   }
 else
      DB2Command myCommand = DB2Context.GetCommand();
      myCommand.CommandText =
                      "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY
                      "
                    + "FROM EMPLOYEE "
                    + "WHERE EMPNO = '" + empID + "'";

      DB2DataReader reader = myCommand.ExecuteReader();

      if (reader.Read())  // Si se encuentra el registro de empleado
      {
         // Obtener nombre y apellidos y salario del empleado
         empName = reader.GetString(0) + " "
         +
         reader.GetString(1) + ". " +
         reader.GetString(2);
         empNameNullInd = 0;
         salary = reader.GetDecimal(3);

         if (bonus == 0)
         {
            if (salary > 75000)
            {
               bonus = salary * (Decimal)0.025;
               bonusNullInd = 0;  // Devolver un valor distinto de NULL
            }
 else
            {
               bonus = salary * (Decimal)0.05;
               bonusNullInd = 0;  // Devolver un valor distinto de NULL
            }
         }
      }
      else  // No se encuentra el empleado
      }
         empName = "";         // Establecer el parámetro de salida
         empNameNullInd = -1;  // Devolver un valor NULL
      }

      reader.Close();
   }
}
```

Ejemplo 4: Procedimiento en C# del estilo de parámetros GENERAL que devuelve un conjunto de resultados:
Este ejemplo muestra lo siguiente:

- Sentencia CREATE PROCEDURE para un procedimiento en C# externo que devuelve un conjunto de resultados
- Código C# para un procedimiento del estilo de parámetros GENERAL que devuelve un conjunto de resultados

Este procedimiento acepta el nombre de una tabla como parámetro. Devuelve un conjunto de resultados que contiene todas las filas de la tabla especificada por el parámetro de entrada. Esto se realiza dejando abierto un `DB2DataReader` para un conjunto de resultados de consulta determinado cuando el procedimiento efectúa la devolución. Específicamente, si no se ejecuta `reader.Close()`, se devolverá el conjunto de resultados.

Tabla 9. Código para crear un procedimiento en C# que devuelve un conjunto de resultados

```
CREATE PROCEDURE ReturnResultSet(IN tableName
VARCHAR(20))
SPECIFIC ReturnResultSet
DYNAMIC RESULT SETS 1
LANGUAGE CLR
PARAMETER STYLE GENERAL
FENCED
PROGRAM TYPE SUB
EXTERNAL NAME
'gwenProc.dll:bizLogic.empOps!ReturnResultSet' ;
```

```
public static void ReturnResultSet(string tableName)
{
   DB2Command myCommand = DB2Context.GetCommand();

   // Establecer la sentencia de SQL a ejecutar y ejecutarla.
   myCommand.CommandText = "SELECT * FROM " + tableName;
   DB2DataReader reader = myCommand.ExecuteReader();

   // El DB2DataReader contiene el resultado de la consulta.
   // Este conjunto de resultados se puede devolver con el procedimiento
   // simplemente NO cerrando el DB2DataReader.
   // Específicamente, NO ejecutar reader.Close();
}
```

Ejemplo 5: Procedimiento en C# del estilo de parámetros SQL que accede a la estructura dbinfo:
Este ejemplo muestra lo siguiente:

- Sentencia CREATE PROCEDURE para un procedimiento que accede a la estructura `dbinfo`
- Código C# para un procedimiento del estilo de parámetros SQL que accede a la estructura `dbinfo`

Para acceder a la estructura `dbinfo`, se debe especificar la cláusula DBINFO en la sentencia CREATE PROCEDURE. No es necesario ningún parámetro para la estructura `dbinfo` en la sentencia CREATE PROCEDURE; no obstante, se debe crear un parámetro para la misma en el código externo de la rutina. Este procedimiento sólo devuelve el valor del nombre de base de datos actual desde el campo `dbname` de la estructura `dbinfo`.

Tabla 10. Código para crear un procedimiento en C# que accede a la estructura dbinfo

```
CREATE PROCEDURE ReturnDbName(OUT dbName VARCHAR(20))
SPECIFIC ReturnDbName
DYNAMIC RESULT SETS 0
LANGUAGE CLR
PARAMETER STYLE SQL
FENCED
DBINFO
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!ReturnDbName'
;
```

```
public static void ReturnDbName(out string dbName,
                                 out Int16  dbNameNullInd,
                                 ref string sqlStateate,
                                     string funcName,
                                     string specName,
                                 ref string sqlMessageText,
                                     sqludf_dbinfo dbinfo)
{
   // Recuperar el nombre de base de datos actual desde la
   // estructura dbinfo y devolverlo.
   // ** Nota ** Los nombres del campo dbinfo son sensibles
   // a las mayúsculas y minúsculas
   dbName = dbinfo.dbname;
   dbNameNullInd = 0;  // Devolver un valor no nulo;

   // Si desea devolver un error definido por el usuario en la
   // SQLCA puede especificar un sqlStateate de 5 dígitos definido
   // por el usuario y el texto de la serie de un mensaje de error.
   // Por ejemplo:
   //
   //   sqlStateate = "ABCDE";
   //   sqlMessageText = "A user-defined error has occured"
   //
   //  DB2 devuelve los valores anteriores al cliente en la
   //  estructura SQLCA. Los valores se utilizan para generar
   //  un error sqlStateate estándar de DB2.
}
```

Ejemplo 6: Procedimiento en C# con el estilo PROGRAM TYPE MAIN:
Este ejemplo muestra lo siguiente:

- Sentencia CREATE PROCEDURE para un procedimiento que utiliza un estilo de programa principal
- Código C# para el estilo de parámetros GENERAL WITH NULLS en la utilización de un estilo de programa MAIN

Para implementar una rutina en un estilo de programa principal, se debe especificar la cláusula PROGRAM TYPE en la sentencia CREATE PROCEDURE con el valor MAIN. Se especifican parámetros en la sentencia CREATE PROCEDURE; no obstante, en la implementación del código, los parámetros se pasan a la rutina en un parámetro entero `argc` y una matriz de parámetros `argv`.

Tabla 11. Código para crear un procedimiento en C# del estilo PROGRAM TYPE MAIN

```
CREATE PROCEDURE MainStyle( IN empID CHAR(6),
                            INOUT bonus Decimal(9,2),
                            OUT empName VARCHAR(60))
SPECIFIC MainStyle
DYNAMIC RESULT SETS 0
LANGUAGE CLR
PARAMETER STYLE GENERAL WITH NULLS
FENCED
PROGRAM TYPE MAIN
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!main' ;
```

Tabla 11. Código para crear un procedimiento en C# del estilo PROGRAM TYPE MAIN (continuación)

```
public static void main(Int32 argc, Object[]
argv)
{
  String empID = (String)argv[0];  // argv[0] tiene nullInd:argv[3]
  Decimal bonus = (Decimal)argv[1]; // argv[1] tiene nullInd:argv[4]
                                    // argv[2] tiene nullInd:argv[5]
  Decimal salary = 0;
  Int16[] NullInds =
  (Int16[])argv[3];

  if ((NullInds[0]) == (Int16)(-1)) // Comprobar si empID es nulo
  {
     NullInds[1] = (Int16)(-1);  // Devolver un valor de bonificación NULL
     argv[1] = (String)"";  // Establecer el parámetro de salida empName
     NullInds[2] = (Int16)(-1);  // Devolver un valor empName NULL
     Return;
  }
 else
     DB2Command myCommand = DB2Context.GetCommand();
     myCommand.CommandText =
                     "SELECT FIRSTNME, MIDINIT, LASTNAME, salary "
                   + "FROM EMPLOYEE "
                   + "WHERE EMPNO = '" + empID + "'";

     DB2DataReader reader = myCommand.ExecuteReader();

     if (reader.Read())  // Si se encuentra el registro de empleado
     {
       // Obtener nombre y apellidos y salario del empleado
       argv[2] = (String) (reader.GetString(0) + " " +
                           reader.GetString(1) + ".
                           " +
                           reader.GetString(2));
       NullInds[2] = (Int16)0;
       salary = reader.GetDecimal(3);

       if (bonus == 0)
       {
          if (salary > 75000)
          {
             argv[1] = (Decimal)(salary * (Decimal)0.025);
             NullInds[1] = (Int16)(0);  // Devolver un valor distinto de NULL
          }
 else
          {
             argv[1] = (Decimal)(salary * (Decimal)0.05);
             NullInds[1] = (Int16)(0);  // Devolver un valor distinto de NULL
          }
       }
     }
     else  // No se encuentra el empleado
     {
        argv[2] = (String)("");         // Establecer el parámetro de salida
        NullInds[2] = (Int16)(-1);      // Devolver un valor NULL
     }

     reader.Close();
   }
}
```

Conceptos relacionados:

- "Rutinas de ejecución en el lenguaje común (CLR)" en la página 116

- "Rutinas en el desarrollo de aplicaciones" en la página 3

Tareas relacionadas:

- "Ejemplos de funciones CLR definidas por el usuario en C#" en la página 152
- "Creación de rutinas CLR" en la página 117
- "Creación de rutinas Common Language Runtime (CLR) .NET" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

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"

Ejemplos de procedimientos CLR en Visual Basic

Una vez comprendidos los conceptos básicos de los procedimientos, también denominados procedimientos almacenados, y los fundamentos de las rutinas de ejecución en el lenguaje común .NET, puede empezar a utilizar procedimientos CLR en sus aplicaciones.

Este tema contiene ejemplos de procedimientos CLR implementados en `Visual Basic`; éstos ilustran los estilos de parámetros soportados, el pase de parámetros, incluida la estructura `dbinfo`, cómo devolver un conjunto de resultados y más información. Para obtener ejemplos de UDF CLR en `Visual Basic`:

- "Ejemplos de funciones CLR definidas por el usuario en Visual Basic" en la página 158

Requisitos previos:

Antes de trabajar con los ejemplos de procedimientos CLR, puede ser conveniente que lea los temas sobre los conceptos siguientes:

- "Rutinas de ejecución en el lenguaje común (CLR)" en la página 116
- "Creación de rutinas CLR" en la página 117
- "Rutinas en el desarrollo de aplicaciones" en la página 3
- Creación de rutinas .NET de ejecución en el lenguaje común (CLR)

Los ejemplos siguientes utilizan una tabla denominada `EMPLOYEE` que está incluida en la base de datos `SAMPLE`.

Procedimiento:

Utilice los ejemplos siguientes como referencias al crear sus propios procedimientos CLR en Visual Basic:

- "Archivo de código externo Visual Basic" en la página 143
- "Ejemplo 1: Procedimiento en Visual Basic del estilo de parámetros GENERAL" en la página 143
- "Ejemplo 2: Procedimiento en Visual Basic del estilo de parámetros GENERAL WITH NULLS" en la página 144
- "Ejemplo 3: Procedimiento en Visual Basic del estilo de parámetros SQL" en la página 146

- "Ejemplo 4: Procedimiento en Visual Basic que devuelve un conjunto de resultados" en la página 148
- "Ejemplo 5: Procedimiento en Visual Basic que accede a la estructura dbinfo" en la página 149
- "Ejemplo 6: Procedimiento en Visual Basic del estilo PROGRAM TYPE MAIN" en la página 150

Archivo de código externo Visual Basic:

Los ejemplos muestran una variedad de implementaciones de procedimientos en Visual Basic. Cada ejemplo se compone de dos partes: la sentencia CREATE PROCEDURE y la implementación en código Visual Basic externo del procedimiento desde el cual se puede crear el ensamblaje asociado.

El archivo fuente en Visual Basic que contiene las implementaciones de procedimientos de los ejemplos siguientes se denomina `gwenVbProc.vb` y tiene el formato siguiente:

Tabla 12. Formato del archivo de código externo Visual Basic

```
using System;
using System.IO;
using IBM.Data.DB2;

Namespace bizLogic

   Class empOps
              ...
     ' Procedimientos en Visual Basic
              ...
   End Class
End Namespace
```

Las inclusiones del archivo se indican al principio del mismo. La inclusión `IBM.Data.DB2` es necesaria si alguno de los procedimientos del archivo contiene SQL. Existe una declaración de espacio de nombres en este archivo y una clase `empOps` que contiene los procedimientos. El uso de espacios de nombres es opcional. Si se utiliza un espacio de nombres, éste debe aparecer en el nombre de vía de acceso de ensamblaje proporcionado en la cláusula EXTERNAL de la sentencia CREATE PROCEDURE.

Es importante tener en cuenta el nombre del archivo, el espacio de nombres y el nombre de la clase, que contiene una implementación de procedimiento determinada. Estos nombres son importantes, ya que la cláusula EXTERNAL de la sentencia CREATE PROCEDURE correspondiente a cada procedimiento debe especificar esta información a fin de que DB2 pueda localizar el ensamblaje y la clase del procedimiento CLR.

Ejemplo 1: Procedimiento en Visual Basic del estilo de parámetros GENERAL:

Este ejemplo muestra lo siguiente:

- Sentencia CREATE PROCEDURE para un procedimiento del estilo de parámetros GENERAL
- Código Visual Basic para un procedimiento del estilo de parámetros GENERAL

Este procedimiento toma un ID de empleado y una cantidad de bonificación actual como entrada. Recupera el nombre y el salario del empleado. Si la cantidad de

bonificación actual es cero, se calcula una nueva bonificación basada en el salario del empleado y se devuelve junto con el nombre y apellidos del empleado. Si no se encuentra el empleado, se devuelve una serie vacía.

Tabla 13. Código para crear un procedimiento en Visual Basic del estilo de parámetros GENERAL

```
  CREATE PROCEDURE SetEmpBonusGEN(IN empId CHAR(6),
                                  INOUT bonus Decimal(9,2),
                                  OUT empName VARCHAR(60))
  SPECIFIC setEmpBonusGEN
  LANGUAGE CLR
PARAMETER STYLE GENERAL
     DYNAMIC RESULT SETS 0
  FENCED
PROGRAM TYPE SUB
  EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!SetEmpBonusGEN'
```

```
    Public Shared Sub SetEmpBonusGEN(ByVal empId As String, _
                                     ByRef bonus As Decimal, _
                                     ByRef empName As String)

       Dim salary As Decimal
       Dim myCommand As DB2Command
       Dim myReader As DB2DataReader

       salary = 0

       myCommand = DB2Context.GetCommand()
       myCommand.CommandText = _
                  "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY " _
                + "FROM EMPLOYEE " _
                + "WHERE EMPNO = '" + empId + "'"
       myReader = myCommand.ExecuteReader()

       If myReader.Read()  ' Si se encuentra el registro de empleado
          ' Obtener nombre y apellidos y salario del empleado
          empName = myReader.GetString(0) + " " _
                  + myReader.GetString(1) + ". " _
                  + myReader.GetString(2)

          salary = myReader.GetDecimal(3)

          If bonus = 0
             If salary > 75000
                bonus = salary * 0.025
             Else
                bonus = salary * 0.05
             End If
          End If
       Else  ' No se encuentra el empleado
          empName = ""  ' Establecer el parámetro de salida
       End If

       myReader.Close()

    End Sub
```

Ejemplo 2: Procedimiento en Visual Basic del estilo de parámetros GENERAL WITH NULLS:
Este ejemplo muestra lo siguiente:

- Sentencia CREATE PROCEDURE para un procedimiento del estilo de parámetros GENERAL WITH NULLS
- Código Visual Basic para un procedimiento del estilo de parámetros GENERAL WITH NULLS

Este procedimiento toma un ID de empleado y una cantidad de bonificación actual como entrada. Si el parámetro de entrada no es nulo, recupera el nombre y salario del empleado. Si la cantidad de bonificación actual es cero, se calcula una nueva bonificación basada en el salario y se devuelve junto con el nombre y apellidos del empleado. Si no se encuentran los datos de empleado, se devuelven un entero y una serie NULL.

Tabla 14. Código para crear un procedimiento en Visual Basic del estilo de parámetros GENERAL WITH NULLS

```
  CREATE PROCEDURE SetEmpBonusGENNULL(IN empId CHAR(6),
                                      INOUT bonus Decimal(9,2),
                                      OUT empName VARCHAR(60))
  SPECIFIC SetEmpBonusGENNULL
  LANGUAGE CLR
  PARAMETER STYLE GENERAL WITH NULLS
      DYNAMIC RESULT SETS 0
  FENCED
PROGRAM TYPE SUB
  EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!SetEmpBonusGENNULL'
```

Tabla 14. Código para crear un procedimiento en Visual Basic del estilo de parámetros GENERAL WITH NULLS (continuación)

```
Public Shared Sub SetEmpBonusGENNULL(ByVal empId As String, _
                                     ByRef bonus As Decimal, _
                                     ByRef empName As String, _
                                     byVal nullInds As Int16())

   Dim salary As Decimal
   Dim myCommand As DB2Command
   Dim myReader As DB2DataReader

   salary = 0

   If nullInds(0) = -1   ' Comprobar si la entrada es nula
      nullInds(1) = -1   ' Devolver un valor de bonificación NULL
      empName = ""       ' Establecer el parámetro de salida
      nullInds(2) = -1   ' Devolver un valor empName NULL
      Return
   Else
      myCommand = DB2Context.GetCommand()
      myCommand.CommandText = _
                "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY " _
              + "FROM EMPLOYEE " _
              + "WHERE EMPNO = '" + empId + "'"

      myReader = myCommand.ExecuteReader()

      If myReader.Read()  ' Si se encuentra el registro de empleado
         ' Obtener nombre y apellidos y salario del empleado
         empName = myReader.GetString(0) + " " _
                 + myReader.GetString(1) + ". " _
                 + myReader.GetString(2)

         salary = myReader.GetDecimal(3)

         If bonus = 0
            If salary > 75000
               bonus = Salary * 0.025
               nullInds(1) = 0 'Devolver un valor distinto de NULL
            Else
               bonus = salary * 0.05
               nullInds(1) = 0 ' Devolver un valor distinto de NULL
            End If
         Else  'No se encuentra el empleado
            empName = ""       ' Establecer el parámetro de salida
            nullInds(2) = -1   ' Devolver un valor NULL
         End If
      End If

      myReader.Close()

   End If

End Sub
```

Ejemplo 3: Procedimiento en Visual Basic del estilo de parámetros SQL:
Este ejemplo muestra lo siguiente:

- Sentencia CREATE PROCEDURE para un procedimiento del estilo de parámetros SQL
- Código Visual Basic para un procedimiento del estilo de parámetros SQL

Este procedimiento toma un ID de empleado y una cantidad de bonificación actual como entrada. Recupera el nombre y el salario del empleado. Si la cantidad de bonificación actual es cero, se calcula una nueva bonificación basada en el salario y

se devuelve junto con el nombre y apellidos del empleado. Si no se encuentra el empleado, se devuelve una serie vacía.

Tabla 15. Código para crear un procedimiento en Visual Basic del estilo de parámetros SQL con parámetros

```
  CREATE PROCEDURE SetEmpBonusSQL(IN empId CHAR(6),
                                  INOUT bonus Decimal(9,2),
                                  OUT empName VARCHAR(60))
  SPECIFIC SetEmpBonusSQL
  LANGUAGE CLR
     PARAMETER STYLE SQL
      DYNAMIC RESULT SETS 0
  FENCED
PROGRAM TYPE SUB
  EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!SetEmpBonusSQL'
```

Tabla 15. Código para crear un procedimiento en Visual Basic del estilo de parámetros SQL con parámetros (continuación)

```
Public Shared Sub SetEmpBonusSQL(byVal empId As String, _
                                 byRef bonus As Decimal, _
                                 byRef empName As String, _
                                 byVal empIdNullInd As Int16, _
                                 byRef bonusNullInd As Int16, _
                                 byRef empNameNullInd As Int16, _
                                 byRef sqlState As String, _
                                 byVal funcName As String, _
                                 byVal specName As String, _
                                 byRef sqlMessageText As String)

   ' Declarar las variables locales del lenguaje principal
   Dim salary As Decimal
   Dim myCommand As DB2Command
   Dim myReader As DB2DataReader

   salary = 0

   If empIdNullInd = -1   ' Comprobar si la entrada es nula
      bonusNullInd = -1   ' Devolver un valor de bonificación NULL
      empName = ""
      empNameNullInd = -1 ' Devolver un valor empName NULL
   Else
     myCommand = DB2Context.GetCommand()
     myCommand.CommandText = _
                "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY " _
              + "FROM EMPLOYEE " _
              + " WHERE EMPNO = '" + empId + "'"

      myReader = myCommand.ExecuteReader()

      If myReader.Read()  ' Si se encuentra el registro de empleado
         '  Obtener nombre y apellidos y salario del empleado
         empName = myReader.GetString(0) + " "
                 + myReader.GetString(1) _
                 + ". " +  myReader.GetString(2)
         empNameNullInd = 0
         salary = myReader.GetDecimal(3)

         If bonus = 0
            If salary > 75000
               bonus = salary * 0.025
               bonusNullInd = 0  ' Devolver un valor distinto de NULL
            Else
               bonus = salary * 0.05
               bonusNullInd = 0  ' Devolver un valor distinto de NULL
            End If
         End If
      Else  ' No se encuentra el empleado
         empName = ""            ' Establecer el parámetro de salida
         empNameNullInd = -1     ' Devolver un valor NULL
      End If

      myReader.Close()
   End If

End Sub
```

Ejemplo 4: Procedimiento en Visual Basic del estilo de parámetros GENERAL que devuelve un conjunto de resultados:
Este ejemplo muestra lo siguiente:

- Sentencia CREATE PROCEDURE para un procedimiento en Visual Basic externo que devuelve un conjunto de resultados

- Código Visual Basic para un procedimiento del estilo de parámetros GENERAL que devuelve un conjunto de resultados

Este procedimiento acepta el nombre de una tabla como parámetro. Devuelve un conjunto de resultados que contiene todas las filas de la tabla especificada por el parámetro de entrada. Esto se realiza dejando abierto un `DB2DataReader` para un conjunto de resultados de consulta determinado cuando el procedimiento efectúa la devolución. Específicamente, si no se ejecuta `reader.Close()`, se devolverá el conjunto de resultados.

Tabla 16. Código para crear un procedimiento en Visual Basic que devuelve un conjunto de resultados

```
CREATE PROCEDURE ReturnResultSet(IN tableName VARCHAR(20))
SPECIFIC ReturnResultSet
DYNAMIC RESULT SETS 1
LANGUAGE CLR
PARAMETER STYLE GENERAL
FENCED
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!ReturnResultSet'
```

```
Public Shared Sub ReturnResultSet(byVal tableName As String)

     Dim myCommand As DB2Command
     Dim myReader As DB2DataReader

     myCommand = DB2Context.GetCommand()

     ' Establecer la sentencia de SQL a ejecutar y ejecutarla.
     myCommand.CommandText = "SELECT * FROM " + tableName
     myReader = myCommand.ExecuteReader()

     ' El DB2DataReader contiene el resultado de la consulta.
     ' Este conjunto de resultados se puede devolver con el procedimiento
     ' simplemente NO cerrando el DB2DataReader.
     ' Específicamente, NO ejecutar reader.Close()

   End Sub
```

Ejemplo 5: Procedimiento en Visual Basic del estilo de parámetros SQL que accede a la estructura dbinfo:
Este ejemplo muestra lo siguiente:

- Sentencia CREATE PROCEDURE para un procedimiento que accede a la estructura `dbinfo`
- Código Visual Basic para un procedimiento del estilo de parámetros SQL que accede a la estructura `dbinfo`

Para acceder a la estructura `dbinfo`, se debe especificar la cláusula DBINFO en la sentencia CREATE PROCEDURE. No es necesario ningún parámetro para la estructura `dbinfo` en la sentencia CREATE PROCEDURE; no obstante, se debe crear un parámetro para la misma en el código externo de la rutina. Este procedimiento sólo devuelve el valor del nombre de base de datos actual desde el campo `dbname` de la estructura `dbinfo`.

Tabla 17. Código para crear un procedimiento en Visual Basic que accede a la estructura dbinfo

```
CREATE PROCEDURE ReturnDbName(OUT dbName VARCHAR(20))
SPECIFIC ReturnDbName
LANGUAGE CLR
PARAMETER STYLE SQL
DBINFO
FENCED
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!ReturnDbName'
```

```
  Public Shared Sub ReturnDbName(byRef dbName As String, _
                                 byRef dbNameNullInd As Int16, _
                                 byRef sqlState As String, _
                                 byVal funcName As String, _
                                 byVal specName As String, _
                                 byRef sqlMessageText As String, _
                                 byVal dbinfo As sqludf_dbinfo)

    ' Recuperar el nombre de base de datos actual desde la
    ' estructura dbinfo y devolverlo.
    dbName = dbinfo.dbname
    dbNameNullInd = 0  ' Devolver un valor no nulo

    ' Si desea devolver un error definido por el usuario en la
    ' SQLCA puede especificar un SQLSTATE de 5 dígitos definido
    ' por el usuario y el texto de la serie de un mensaje de error.
    ' Por ejemplo:
    '
    ' sqlState = "ABCDE"
    ' msg_token = "A user-defined error has occured"
    '
    ' DB2 devolverá estos valores en la SQLCA. Aparecerá
    ' con el formato de un error sqlState normal de
    ' DB2.
  End Sub
```

Ejemplo 6: Procedimiento en Visual Basic con el estilo PROGRAM TYPE MAIN:

Este ejemplo muestra lo siguiente:

- Sentencia CREATE PROCEDURE para un procedimiento que utiliza un estilo de programa principal
- Código Visual Basic para el estilo de parámetros GENERAL WITH NULLS en la utilización de un estilo de programa MAIN

Para implementar una rutina en un estilo de programa principal, se debe especificar la cláusula PROGRAM TYPE en la sentencia CREATE PROCEDURE con el valor MAIN. Se especifican parámetros en la sentencia CREATE PROCEDURE; no obstante, en la implementación del código, los parámetros se pasan a la rutina en un parámetro entero `argc` y una matriz de parámetros `argv`.

Tabla 18. Código para crear un procedimiento en Visual Basic del estilo PROGRAM TYPE MAIN

```
CREATE PROCEDURE MainStyle(IN empId CHAR(6),
                           INOUT bonus Decimal(9,2),
                           OUT empName VARCHAR(60))
SPECIFIC mainStyle
DYNAMIC RESULT SETS 0
LANGUAGE CLR
PARAMETER STYLE GENERAL WITH NULLS
FENCED
PROGRAM TYPE MAIN
EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!Main'
```

Tabla 18. Código para crear un procedimiento en Visual Basic del estilo PROGRAM TYPE MAIN (continuación)

```
Public Shared Sub Main( byVal argc As Int32, _
                        byVal argv As Object())

   Dim myCommand As DB2Command
   Dim myReader As DB2DataReader
   Dim empId As String
   Dim bonus As Decimal
   Dim salary As Decimal
   Dim nullInds As Int16()

   empId = argv(0)  ' argv[0] (IN)    nullInd = argv[3]
   bonus = argv(1)  ' argv[1] (INOUT) nullInd = argv[4]
                    ' argv[2] (OUT)   nullInd = argv[5]
   salary = 0
   nullInds = argv(3)

   If nullInds(0) = -1     ' Comprobar si la entrada empId es nula
      nullInds(1) = -1     ' Devolver un valor de bonificación NULL
      argv(1) = ""         ' Establecer el parámetro de salida empName
      nullInds(2) = -1     ' Devolver un valor empName NULL
      Return
   Else
      ' Si el empleado existe y la bonificación actual es 0,
      ' calcular una nueva bonificación basándose en el salario
      ' del empleado. Devolver nombre y nueva bonificación de éste
      myCommand = DB2Context.GetCommand()
      myCommand.CommandText = _
                "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY " _
              + " FROM EMPLOYEE " _
              + " WHERE EMPNO = '" + empId + "'"

      myReader = myCommand.ExecuteReader()


      If myReader.Read()  ' Si se encuentra el registro de empleado
         ' Obtener nombre y apellidos y salario del empleado
         argv(2) = myReader.GetString(0) + " " _
                 + myReader.GetString(1) + ". " _
                 + myReader.GetString(2)
         nullInds(2) = 0
         salary = myReader.GetDecimal(3)

         If bonus = 0
            If salary > 75000
               argv(1) = salary * 0.025
               nullInds(1) = 0  ' Devolver un valor distinto de NULL
            Else
               argv(1) = Salary * 0.05
               nullInds(1) = 0  ' Devolver un valor distinto de NULL
            End If
         End If
      Else  ' No se encuentra el empleado
         argv(2) = ""      ' Establecer el parámetro de salida
         nullInds(2) = -1  ' Devolver un valor NULL
      End If

      myReader.Close()
   End If

End Sub
```

Conceptos relacionados:

- "Rutinas de ejecución en el lenguaje común (CLR)" en la página 116

- "Rutinas en el desarrollo de aplicaciones" en la página 3

Tareas relacionadas:
- "Ejemplos de funciones CLR definidas por el usuario en Visual Basic" en la página 158
- "Creación de rutinas CLR" en la página 117
- "Creación de rutinas Common Language Runtime (CLR) .NET" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Ejemplos de funciones CLR definidas por el usuario en C#

Una vez comprendidos los conceptos básicos de las funciones definidas por el usuario (UDF) y los fundamentos de las rutinas CLR, puede empezar a explotar las UDF CLR en sus aplicaciones y en el entorno de bases de datos. Este tema contiene algunos ejemplos de UDF CLR para poder empezar. Si desea obtener ejemplos de procedimientos CLR en C#:
- "Ejemplos de procedimientos CLR en C#" en la página 130

Requisitos previos:

Antes de trabajar con los ejemplos de UDF CLR, puede ser conveniente que lea los temas sobre los conceptos siguientes:
- "Rutinas de ejecución en el lenguaje común (CLR)" en la página 116
- "Creación de rutinas CLR" en la página 117
- "Funciones escalares definidas por el usuario" en la página 15
- "Funciones escalares definidas por el usuario" en la página 17
- Creación de rutinas .NET de ejecución en el lenguaje común (CLR)

Los ejemplos siguientes utilizan una tabla denominada `EMPLOYEE` que está incluida en la base de datos `SAMPLE`.

Procedimiento:

Utilice los ejemplos siguientes como referencias al crear sus propias UDF CLR en C#:
- "Archivo de código externo C#"
- "Ejemplo 1: Función de tabla en C# del estilo de parámetros SQL" en la página 153
- "Ejemplo 2: Función escalar en C# del estilo de parámetros SQL" en la página 156

Archivo de código externo C#:

Los ejemplos siguientes muestran una variedad de implementaciones de UDF en C#. La sentencia CREATE FUNCTION se proporciona para cada UDF con el código fuente C# correspondiente desde el cual se puede crear el ensamblaje asociado. El archivo fuente en C# que contiene las declaraciones de funciones utilizadas en los ejemplos siguientes se denomina `gwenUDF.cs` y tiene el formato siguiente:

Tabla 19. Formato del archivo de código externo C#

```
using System;
using System.IO;
using IBM.Data.DB2;

namespace bizLogic
{
   ...
   // Definiciones de clases que contienen declaraciones de UDF
   // y cualquier definición de clase de soporte
   ...
}
```

Las declaraciones de funciones deben estar incluidas en una clase dentro de un archivo de C#. El uso de espacios de nombres es opcional. Si se utiliza un espacio de nombres, éste debe aparecer en el nombre de vía de acceso de ensamblaje proporcionado en la cláusula EXTERNAL de la sentencia CREATE PROCEDURE. La inclusión de `IBM.Data.DB2.` es necesaria si la función contiene SQL.

Ejemplo 1: Función de tabla en C# del estilo de parámetros SQL:
Este ejemplo muestra lo siguiente:

- Sentencia CREATE FUNCTION para una función de tabla del estilo de parámetros SQL
- Código C# para una función de tabla del estilo de parámetros SQL

Esta función de tabla devuelve una tabla que contiene filas de los datos de empleado que se han creado a partir de una matriz de datos. Existen dos clases asociadas con este ejemplo. La clase `person` representa los empleados y la clase `empOps` contiene la UDF de tabla de rutina que utiliza la clase `person`. La información sobre el salario de los empleados se actualiza basándose en el valor de un parámetro de entrada. La matriz de datos de este ejemplo se crea dentro de la propia función de tabla en la primera llamada de la función de tabla. Dicha matriz también se podría haber creado leyendo datos de un archivo de texto del sistema de archivos. Los valores de datos de la matriz se graban en un área reutilizable para que sea posible acceder a los datos en llamadas subsiguientes de la función de tabla.

En cada llamada de la función de tabla, se lee un registro de la matriz y se genera una fila en la tabla devuelta por la función. La fila se genera en la tabla estableciendo los parámetros de salida de la función de tabla en los valores de fila deseados. Después de que se produzca la llamada final de la función de tabla, se devuelve la tabla de las filas generadas.

Tabla 20. Código para crear una función de tabla en C# del estilo de parámetros SQL

```
CREATE FUNCTION tableUDF(double)
RETURNS TABLE (name varchar(20),
               job varchar(20),
               salary double)
EXTERNAL NAME 'gwenUDF.dll:bizLogic.empOps!tableUDF'
LANGUAGE CLR
PARAMETER STYLE SQL
     NOT DETERMINISTIC
FENCED
SCRATCHPAD 10
FINAL CALL
DISALLOW PARALLEL
    NO DBINFO
```

Tabla 20. Código para crear una función de tabla en C# del estilo de parámetros SQL (continuación)

```
// La clase Person es una clase de soporte para
// la función de tabla UDF, tableUDF, siguiente.
class Person
{
     private String name;
     private String position;
     private Int32 salary;

     public Person(String newName, String newPosition, Int32
     newSalary)
     {
        this.name = newName;
        this.position = newPosition;
        this.salary = newSalary;
     }

     public String getName()
     {
        return this.name;
     }

     public String getPosition()
     {
        return this.position;
     }

     public Int32 getSalary()
     {
        return this.salary;
     }
 }
```

Tabla 20. Código para crear una función de tabla en C# del estilo de parámetros SQL (continuación)

```
 class empOps
{
  {
   public static void TableUDF( Double factor, out String name,
                     out String position, out Double salary,
                     Int16 factorNullInd, out Int16 nameNullInd,
                     out Int16 positionNullInd, out Int16 salaryNullInd,
                     ref String sqlState, String funcName,
                     String specName, ref String sqlMessageText,
                     Byte[] scratchPad, Int32 callType)
   {

      Int16 intRow = 0;

      // Crear una matriz de información del tipo Person
      Person[] Staff = new
      Person[3];
      Staff[0] = new Person("Gwen", "Developer", 10000);
      Staff[1] = new Person("Andrew", "Developer", 20000);
      Staff[2] = new Person("Liu", "Team Leader", 30000);

      salary = 0;
      name = position = "";
      nameNullInd = positionNullInd = salaryNullInd = -1;

      switch(callType)
      {
         case (-2):  // Case SQLUDF_TF_FIRST:
           break;

         case (-1):  // Case SQLUDF_TF_OPEN:
           intRow = 1;
           scratchPad[0] = (Byte)intRow;  // Grabar en área reutilizable
           break;
         case (0):   // Case SQLUDF_TF_FETCH:
             intRow = (Int16)scratchPad[0];
           if (intRow > Staff.Length)
           {
              sqlState = "02000";  // Devolver un error SQLSTATE
           }
  else
           {
              // Generar una fila en la tabla de salida
              // basada en los datos de la matriz Staff.
              name =
              Staff[intRow-1].getName();
              position = Staff[intRow-1].getPosition();
              salary = (Staff[intRow-1].getSalary[]] * factor;
              nameNullInd = 0;
              positionNullInd = 0;
              salaryNullInd = 0;
           }
           intRow++;
           scratchPad[0] = (Byte)intRow;  // Grabar área reutilizable
           break;

         case (1):   // Case SQLUDF_TF_CLOSE:
           break;

         case (2):   // Case SQLUDF_TF_FINAL:
           break;
      }
   }
}
```

Ejemplo 2: Función escalar en C# del estilo de parámetros SQL:
Este ejemplo muestra lo siguiente:

- Sentencia CREATE FUNCTION para una función escalar del estilo de parámetros SQL
- Código C# para una función escalar del estilo de parámetros SQL

Esta función escalar devuelve un solo valor de cuenta para cada valor de entrada sobre el que actúa. Para un valor de entrada situado en la n[a] posición del conjunto de valores de entrada, el valor escalar de salida es el valor n. En cada llamada de la función escalar, donde una llamada está asociada con cada fila o valor del conjunto de filas o valores de entrada, la cuenta aumenta en uno y se devuelve el valor actual de la cuenta. Luego, la cuenta se guarda en el almacenamiento intermedio de memoria del área reutilizable para mantener el valor de cuenta entre cada llamada de la función escalar.

Esta función escalar se puede invocar fácilmente si, por ejemplo, se dispone de una tabla definida del modo siguiente:

```
CREATE TABLE T (i1 INTEGER);
INSERT INTO T VALUES 12, 45, 16, 99;
```

Se puede utilizar una consulta simple como la siguiente para invocar la función escalar:

```
SELECT countUp(i1) as count, i1 FROM T;
```

La salida de una consulta como la indicada sería:

```
COUNT           I1
-----------     ----------
1               12
2               45
3               16
4               99
```

Esta UDF escalar es bastante simple. En lugar de devolver sólo la cuenta de las filas, puede utilizar una función escalar que formatee los datos de una columna existente. Por ejemplo, puede añadir una serie a cada valor de una columna de direcciones, puede crear una serie compleja a partir de una cadena de series de entrada o puede efectuar un cálculo matemático complejo con un conjunto de datos donde deberá almacenar un resultado intermedio.

Tabla 21. Código para crear una función escalar en C# del estilo de parámetros SQL

```
CREATE FUNCTION countUp(INTEGER)
RETURNS INTEGER
LANGUAGE CLR
PARAMETER STYLE SQL
FENCED
SCRATCHPAD 10
FINAL CALL
VARIANT
NO SQL
EXTERNAL NAME 'gwenUDF.dll:bizLogic.empOps!CountUp' ;
```

Tabla 21. Código para crear una función escalar en C# del estilo de parámetros SQL (continuación)

```
 class empOps
{
   public static void CountUp(     Int32 input,
                                out Int32 outCounter,
                                    Int16 inputNullInd,
                                out Int16 outCounterNullInd,
                                ref String sqlState,
                                    String funcName,
                                    String specName,
                                ref String sqlMessageText,
                                     Byte[] scratchPad,
                                    Int32 callType)
   {
      Int32 counter = 1;
      switch(callType)
      {
         case -1: // case SQLUDF_FIRST_CALL
           scratchPad[0] = (Byte)counter;
           outCounter = counter;
           outCounterNullInd = 0;
           break;
         case 0:  // case SQLUDF_NORMAL_CALL:
           counter = (Int32)scratchPad[0];
           counter = counter + 1;
           outCounter = counter;
           outCounterNullInd = 0;
           scratchPad[0] =
           (Byte)counter;
           break;
         case 1:  // case SQLUDF_FINAL_CALL:
           counter =
           (Int32)scratchPad[0];
           outCounter = counter;
           outCounterNullInd = 0;
           break;
         default: // Nunca se debe entrar aquí
                  // * Necesario para que durante la compilación
                  //   se establezca siempre el parámetro de salida outCounter *
           outCounter = (Int32)(0);
           outCounterNullInd = -1;
           sqlState="ABCDE";
           sqlMessageText = "Should not get here: Default
           case!";
           break;
      }
   }
}
```

Conceptos relacionados:

- "Rutinas de ejecución en el lenguaje común (CLR)" en la página 116
- "Funciones escalares definidas por el usuario" en la página 15
- "Funciones escalares definidas por el usuario" en la página 17

Tareas relacionadas:

- "Ejemplos de procedimientos CLR en C#" en la página 130
- "Creación de rutinas CLR" en la página 117
- "Creación de rutinas Common Language Runtime (CLR) .NET" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

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"

Ejemplos de funciones CLR definidas por el usuario en Visual Basic

Una vez comprendidos los conceptos básicos de las funciones definidas por el usuario (UDF) y los fundamentos de las rutinas CLR, puede empezar a explotar las UDF CLR en sus aplicaciones y en el entorno de bases de datos. Este tema contiene algunos ejemplos de UDF CLR para poder empezar. Si desea obtener ejemplos de procedimientos CLR en Visual Basic:

- "Ejemplos de procedimientos CLR en Visual Basic" en la página 142

Requisitos previos:

Antes de trabajar con los ejemplos de UDF CLR, puede ser conveniente que lea los temas sobre los conceptos siguientes:

- "Rutinas de ejecución en el lenguaje común (CLR)" en la página 116
- "Creación de rutinas CLR" en la página 117
- "Funciones escalares definidas por el usuario" en la página 15
- "Funciones escalares definidas por el usuario" en la página 17
- Creación de rutinas .NET de ejecución en el lenguaje común (CLR)

Los ejemplos siguientes utilizan una tabla denominada `EMPLOYEE` que está incluida en la base de datos `SAMPLE`.

Procedimiento:

Utilice los ejemplos siguientes como referencias al crear sus propias UDF CLR en Visual Basic:

- "Archivo de código externo Visual Basic"
- "Ejemplo 1: Función de tabla en Visual Basic del estilo de parámetros SQL" en la página 159
- "Ejemplo 2: Función escalar en Visual Basic del estilo de parámetros SQL" en la página 161

Archivo de código externo Visual Basic:

Los ejemplos siguientes muestran una variedad de implementaciones de UDF en Visual Basic. La sentencia CREATE FUNCTION se proporciona para cada UDF con el código fuente Visual Basic correspondiente desde el cual se puede crear el ensamblaje asociado. El archivo fuente en Visual Basic que contiene las declaraciones de funciones utilizadas en los ejemplos siguientes se denomina `gwenVbUDF.cs` y tiene el formato siguiente:

Tabla 22. Formato del archivo de código externo Visual Basic

```
  using System;
  using System.IO;
  using IBM.Data.DB2;

  Namespace bizLogic

     ...
     ' Definiciones de clases que contienen declaraciones de UDF
     ' y cualquier definición de clase de soporte
     ...

End Namespace
```

Las declaraciones de funciones deben estar incluidas en una clase dentro de un archivo de Visual Basic. El uso de espacios de nombres es opcional. Si se utiliza un espacio de nombres, éste debe aparecer en el nombre de vía de acceso de ensamblaje proporcionado en la cláusula EXTERNAL de la sentencia CREATE PROCEDURE. La inclusión de `IBM.Data.DB2.` es necesaria si la función contiene SQL.

Ejemplo 1: Función de tabla en Visual Basic del estilo de parámetros SQL: Este ejemplo muestra lo siguiente:

- Sentencia CREATE FUNCTION para una función de tabla del estilo de parámetros SQL
- Código Visual Basic para una función de tabla del estilo de parámetros SQL

Esta función de tabla devuelve una tabla que contiene filas de los datos de empleado que se han creado a partir de una matriz de datos. Existen dos clases asociadas con este ejemplo. La clase `person` representa los empleados y la clase `empOps` contiene la UDF de tabla de rutina que utiliza la clase `person`. La información sobre el salario de los empleados se actualiza basándose en el valor de un parámetro de entrada. La matriz de datos de este ejemplo se crea dentro de la propia función de tabla en la primera llamada de la función de tabla. Dicha matriz también se podría haber creado leyendo datos de un archivo de texto del sistema de archivos. Los valores de datos de la matriz se graban en un área reutilizable para que sea posible acceder a los datos en llamadas subsiguientes de la función de tabla.

En cada llamada de la función de tabla, se lee un registro de la matriz y se genera una fila en la tabla devuelta por la función. La fila se genera en la tabla estableciendo los parámetros de salida de la función de tabla en los valores de fila deseados. Después de que se produzca la llamada final de la función de tabla, se devuelve la tabla de las filas generadas.

Tabla 23. Código para crear una función de tabla en Visual Basic del estilo de parámetros SQL

```
CREATE FUNCTION TableUDF(double)
RETURNS TABLE (name varchar(20),
               job varchar(20),
               salary double)
EXTERNAL NAME 'gwenVbUDF.dll:bizLogic.empOps!TableUDF'
LANGUAGE CLR
PARAMETER STYLE SQL
NOT DETERMINISTIC
FENCED
SCRATCHPAD 10
FINAL CALL
DISALLOW PARALLEL
NO DBINFO
```

```
Class Person
' La clase Person es una clase de soporte para
' la función de tabla UDF, tableUDF, siguiente.

  Private name As String
  Private position As String
  Private salary As Int32

  Public Sub New(ByVal newName As String, _
                 ByVal newPosition As String, _
                 ByVal newSalary As Int32)

    name = newName
    position = newPosition
    salary = newSalary
  End Sub

  Public Property GetName() As String
    Get
      Return name
    End Get

    Set (ByVal value As String)
      name = value
    End Set
  End Property

  Public Property GetPosition() As String
    Get
      Return position
    End Get

    Set (ByVal value As String)
      position = value
    End Set
  End Property

  Public Property GetSalary() As Int32
    Get
      Return salary
    End Get

    Set (ByVal value As Int32)
      salary = value
    End Set
  End Property

End Class
```

Tabla 23. Código para crear una función de tabla en Visual Basic del estilo de parámetros SQL (continuación)

```
Class empOps

  Public Shared Sub TableUDF(byVal factor as Double, _
                             byRef name As String, _
                             byRef position As String, _
                             byRef salary As Double, _
                             byVal factorNullInd As Int16, _
                             byRef nameNullInd As Int16, _
                             byRef positionNullInd As Int16, _
                             byRef salaryNullInd As Int16, _
                             byRef sqlState As String, _
                             byVal funcName As String, _
                             byVal specName As String, _
                             byRef sqlMessageText As String, _
                             byVal scratchPad As Byte(), _
                             byVal callType As Int32)

    Dim intRow As Int16

    intRow = 0

    ' Crear una matriz de información del tipo Person
    Dim staff(2) As Person
    staff(0) = New Person("Gwen", "Developer", 10000)
    staff(1) = New Person("Andrew", "Developer", 20000)
    staff(2) = New Person("Liu", "Team Leader", 30000)

    ' Inicializar valores de parámetro de salida e indicadores NULL
    salary = 0
    name = position = ""
    nameNullInd = positionNullInd = salaryNullInd = -1

    Select callType
       Case -2   ' Case SQLUDF_TF_FIRST:
       Case -1   ' Case SQLUDF_TF_OPEN:
         intRow = 1
         scratchPad(0) = intRow  ' Grabar en área reutilizable
       Case 0    ' Case SQLUDF_TF_FETCH:
         intRow = scratchPad(0)
         If intRow > staff.Length
            sqlState = "02000"  ' Devolver un error SQLSTATE
         Else
            ' Generar una fila en la tabla de salida
            ' basada en los datos de la matriz staff.
            name = staff(intRow).GetName()
            position = staff(intRow).GetPosition()
            salary = (staff(intRow).GetSalary()) * factor
            nameNullInd = 0
            positionNullInd = 0
            salaryNullInd = 0
         End If
         intRow = intRow + 1
         scratchPad(0) = intRow  ' Grabar área reutilizable

       Case 1    ' Case SQLUDF_TF_CLOSE:

       Case 2    ' Case SQLUDF_TF_FINAL:
    End Select

  End Sub

End Class
```

Ejemplo 2: Función escalar en Visual Basic del estilo de parámetros SQL:

Este ejemplo muestra lo siguiente:

- Sentencia CREATE FUNCTION para una función escalar del estilo de parámetros SQL
- Código Visual Basic para una función escalar del estilo de parámetros SQL

Esta función escalar devuelve un solo valor de cuenta para cada valor de entrada sobre el que actúa. Para un valor de entrada situado en la nª posición del conjunto de valores de entrada, el valor escalar de salida es el valor n. En cada llamada de la función escalar, donde una llamada está asociada con cada fila o valor del conjunto de filas o valores de entrada, la cuenta aumenta en uno y se devuelve el valor actual de la cuenta. Luego, la cuenta se guarda en el almacenamiento intermedio de memoria del área reutilizable para mantener el valor de cuenta entre cada llamada de la función escalar.

Esta función escalar se puede invocar fácilmente si, por ejemplo, se dispone de una tabla definida del modo siguiente:

```
CREATE TABLE T (i1 INTEGER);
INSERT INTO T VALUES 12, 45, 16, 99;
```

Se puede utilizar una consulta simple como la siguiente para invocar la función escalar:

```
SELECT my_count(i1) as count, i1 FROM T;
```

La salida de una consulta como la indicada sería:

```
COUNT           I1
-----------     ----------
1               12
2               45
3               16
4               99
```

Esta UDF escalar es bastante simple. En lugar de devolver sólo la cuenta de las filas, puede utilizar una función escalar que formatee los datos de una columna existente. Por ejemplo, puede añadir una serie a cada valor de una columna de direcciones, puede crear una serie compleja a partir de una cadena de series de entrada o puede efectuar un cálculo matemático complejo con un conjunto de datos donde deberá almacenar un resultado intermedio.

Tabla 24. Código para crear una función escalar en Visual Basic del estilo de parámetros SQL

```
  CREATE FUNCTION mycount(INTEGER)
RETURNS INTEGER
  LANGUAGE CLR
     PARAMETER STYLE SQL
  FENCED
SCRATCHPAD 10
  FINAL CALL
VARIANT
     NO SQL
  EXTERNAL NAME 'gwenUDF.dll:bizLogic.empOps!CountUp';
```

Tabla 24. Código para crear una función escalar en Visual Basic del estilo de parámetros SQL (continuación)

```
Class empOps
  Public Shared Sub CountUp(byVal input As Int32, _
                            byRef outCounter As Int32, _
                            byVal nullIndInput As Int16, _
                            byRef nullIndOutCounter As Int16, _
                            byRef sqlState As String, _
                            byVal qualName As String, _
                            byVal specName As String, _
                            byRef sqlMessageText As String, _
                            byVal scratchPad As Byte(), _
                            byVal callType As Int32)

     Dim counter As Int32
     counter = 1

     Select callType
        case -1           ' case SQLUDF_TF_OPEN_CALL
           scratchPad(0) = counter
           outCounter = counter
           nullIndOutCounter = 0
        case 0              'case SQLUDF_TF_FETCH_CALL:
           counter = scratchPad(0)
           counter = counter + 1
           outCounter = counter
           nullIndOutCounter = 0
           scratchPad(0) = counter
        case 1             'case SQLUDF_CLOSE_CALL:
           counter = scratchPad(0)
           outCounter = counter
           nullIndOutCounter = 0
        case Else          ' Nunca se debe entrar aquí
           ' Estos casos no se producirán por las razones siguientes:
           ' Case -2  (SQLUDF_TF_FIRST)     ->No hay FINAL CALL en sent. CREATE
           ' Case 2   (SQLUDF_TF_FINAL)     ->No hay FINAL CALL en sent. CREATE
           ' Case 255 (SQLUDF_TF_FINAL_CRA) ->No se utiliza SQL en la función
           '
           ' * Nota*
           ' ---------
           ' Else es necesario para que durante la compilación
           ' se establezca siempre el parámetro de salida outCounter *
           outCounter = 0
           nullIndOutCounter = -1
     End Select
  End Sub

End Class
```

Conceptos relacionados:

- "Rutinas de ejecución en el lenguaje común (CLR)" en la página 116
- "Funciones escalares definidas por el usuario" en la página 15
- "Funciones escalares definidas por el usuario" en la página 17

Tareas relacionadas:

- "Ejemplos de procedimientos CLR en Visual Basic" en la página 142
- "Creación de rutinas CLR" en la página 117
- "Creación de rutinas Common Language Runtime (CLR) .NET" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Rutinas C/C++

Los apartados siguientes describen cómo escribir rutinas C o C++.

Rutinas C/C++

Cuando se desarrollan rutinas en C o C++, es muy recomendable registrarlas utilizando la cláusula PARAMETER STYLE SQL en la sentencia CREATE. También es recomendable utilizar el archivo de inclusión sqludf.h. Éste contiene estructuras, definiciones y valores que son de utilidad para escribir tanto UDF como procedimientos almacenados.

UDF y métodos C/C++:

La signatura en C/C++ de las UDF y los métodos con PARAMETER STYLE SQL sigue este formato:

```
SQL_API_RC SQL_API_FN nombre-función ( argumentos-SQL,
                                       SQL-argument-inds,
                                       SQLUDF_TRAIL_ARGS )
```

`SQL_API_RC SQL_API_FN`
: `SQL_API_RC` y `SQL_API_FN` son macros que especifican el tipo de retorno y el convenio de llamada para una función C/C++, los cuales pueden ser distintos según los sistemas operativos soportados. Se declaran en sqlsystm.h. Esta macro es necesaria cuando se escriben rutinas C/C++.

nombre-función
: El nombre de la función C/C++. Durante el registro de la rutina, se especifica este valor junto con el nombre de biblioteca en la cláusula EXTERNAL NAME de la sentencia CREATE PROCEDURE. En las rutinas C++, el compilador C++ aplica la decoración de tipos al nombre de punto de entrada. Hay que especificar el nombre decorado con tipos en la cláusula EXTERNAL NAME o bien se debe definir el punto de entrada como `extern "C"` en el código del usuario.

argumentos-SQL
: Corresponde a la lista de parámetros de entrada de la sentencia CREATE de la rutina.

ind-argumentos-SQL
: Para cada argumento-SQL, existe una variable de indicador. Defina cada indicador con la definición de tipo SQLUDF_NULLIND de sqludf.h.

`SQLUDF_TRAIL_ARGS`
: Una macro definida en sqludf.h que define los argumentos de cola para una rutina. Se incluyen punteros al SQLSTATE, el nombre completamente calificado de la función, el nombre específico de la función y texto de mensaje. Si la UDF se registra con SCRATCHPAD y FINAL CALL, utilice la macro `SQLUDF_TAIL_ARGS_ALL`. Además de los argumentos incluidos en `SQLUDF_TRAIL_ARGS`, ésta contiene punteros al área reutilizable y al tipo de llamada.

A continuación, se muestra un ejemplo de una UDF escrita en C/C++ que devuelve el producto de sus dos argumentos de entrada:

```
SQL_API_RC SQL_API_FN product ( SQLUDF_DOUBLE *in1,
                                SQLUDF_DOUBLE *in2,
                                SQLUDF_DOUBLE *outProduct,
                                SQLUDF_NULLIND *in1NullInd,
                                SQLUDF_NULLIND *in2NullInd,
```

```
                                SQLUDF_NULLIND *productNullInd,
                                SQLUDF_TRAIL_ARGS )
{
  *outProduct = (*in1) * (*in2);

  return (0);
}
```

La sentencia CREATE FUNCTION correspondiente a esta UDF es la siguiente:

```
CREATE FUNCTION product( DOUBLE in1, DOUBLE in2 )
RETURNS DOUBLE
  LANGUAGE c
  PARAMETER STYLE sql
  NO SQL
  FENCED THREADSAFE
  DETERMINISTIC
  RETURNS NULL ON NULL INPUT
  NO EXTERNAL ACTION
  EXTERNAL NAME 'c_rtns!product'
```

La sentencia anterior supone que la función C/C++ se halla en una biblioteca denominada c_rtns.

Procedimientos almacenados C/C++:

La signatura en C/C++ de los procedimientos almacenados con PARAMETER STYLE SQL sigue este formato:

```
SQL_API_RC SQL_API_FN nombre-función ( argumentos-SQL,
                                       SQL-argument-inds,
                                       sqlstate,
                                       nombre-rutina,
                                       nombre-específico,
                                       mensaje-diagnóstico )
```

SQL_API_RC SQL_API_FN
: SQL_API_RC y SQL_API_FN son macros que especifican el tipo de retorno y el convenio de llamada para una función C/C++, los cuales pueden ser distintos según los sistemas operativos soportados. Se declaran en sqlsystm.h. Esta macro es necesaria cuando se escriben rutinas C/C++.

nombre-función
: El nombre de la función C/C++. Durante el registro de la rutina, se especifica este valor junto con el nombre de biblioteca en la cláusula EXTERNAL NAME de la sentencia CREATE PROCEDURE. En las rutinas C++, el compilador C++ aplica la decoración de tipos al nombre de punto de entrada. Hay que especificar el nombre decorado con tipos en la cláusula EXTERNAL NAME o bien se debe definir el punto de entrada como "C" externo en el código del usuario.

argumentos-SQL
: Corresponde a la lista de parámetros de entrada de la sentencia CREATE PROCEDURE. Los parámetros de modalidad OUT o INOUT se pasan como matrices de un solo elemento.

sqlstate Lo utiliza la rutina para indicar condiciones de aviso o de error.

nombre-rutina
: El nombre calificado de la función. DB2® genera este valor, que pasa a la rutina con el formato esquema.rutina. Este valor se corresponde con las columnas ROUTINESCHEMA y ROUTINENAME de la vista SYSCAT.ROUTINES.

nombre-específico
El nombre específico de la función. DB2 genera este valor, que pasa a la rutina. Este valor se corresponde con la columna SPECIFICNAME de la vista SYSCAT.ROUTINES.

mensaje-diagnóstico
Lo utiliza la rutina para devolver texto de mensaje a la aplicación o rutina que realiza la invocación.

Nota: A diferencia de la signatura de función presentada en el apartado sobre las UDF y los métodos C/C++, la signatura de función presentada para los procedimientos almacenados C/C++ no utiliza macros declaradas en sqludf.h. Sin embargo, es posible escribir procedimientos almacenados C/C++ con las macros de sqludf.h. A la inversa, también es posible escribir UDF y métodos C/C++ sin las macros de sqludf.h.

A continuación, se muestra un ejemplo de un procedimiento almacenado escrito en C/C++ que acepta un parámetro de entrada y luego devuelve un parámetro de salida y un conjunto de resultados:

```
SQL_API_RC SQL_API_FN cstp ( sqlint16 *inParm,
                             double *outParm,
                             sqlint16 *inParmNullInd,
                             sqlint16 *outParmNullInd,
                             char sqlst[6],
                             char qualname[28],
                             char specname[19],
                             char diagmsg[71] )
{
  EXEC SQL INCLUDE SQLCA;

  EXEC SQL BEGIN DECLARE SECTION;
    sqlint16 sql_inParm;
  EXEC SQL END DECLARE SECTION;


  sql_inParm = *inParm;

  EXEC SQL DECLARE cur1 CURSOR FOR
    SELECT value
    FROM table01
    WHERE index = :sql_inParm;

  *outParm = (*inParm) + 1;

  EXEC SQL OPEN cur1;

  return (0);
}
```

La sentencia CREATE PROCEDURE correspondiente a este procedimiento almacenado es la siguiente:

```
CREATE PROCEDURE cproc( IN inParm INT, OUT outParm INT )
  LANGUAGE c
  PARAMETER STYLE sql
  DYNAMIC RESULT SETS 1
  FENCED THREADSAFE
  RETURNS NULL ON NULL INPUT
  EXTERNAL NAME 'c_rtns!cstp'
```

La sentencia anterior supone que la función C/C++ se halla en una biblioteca denominada c_rtns.

Nota: Al registrar una rutina C o C++ en los sistemas operativos Windows®, tome la precaución siguiente cuando identifique un cuerpo de rutina en la cláusula EXTERNAL NAME de la sentencia CREATE. Si utiliza un ID de vía de acceso absoluto para identificar el cuerpo de la rutina, debe añadir la extensión .dll. Por ejemplo:

```
CREATE PROCEDURE getSalary( IN inParm INT, OUT outParm INT )
  LANGUAGE c
  PARAMETER STYLE sql
  DYNAMIC RESULT SETS 1
  FENCED THREADSAFE
  RETURNS NULL ON NULL INPUT
  EXTERNAL NAME 'd:\mylib\myfunc.dll'
```

Conceptos relacionados:

- "Instancias del gestor de base de datos" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Archivos de exportación de AIX para rutinas" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Rutinas de AIX y la sentencia CREATE" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Archivo de inclusión para rutinas C/C++ (sqludf.h)" en la página 168
- "Manejo de tipos de datos de SQL en rutinas C/C++" en la página 171

Tareas relacionadas:

- "Creación de rutinas C para UNIX" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de rutinas C++ para UNIX" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de rutinas C/C++ en Windows" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Información relacionada:

- "Sentencia CREATE FUNCTION" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE PROCEDURE" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE TYPE (Estructurado)" en la publicación *Consulta de SQL, Volumen 2*
- "Ejemplos de C" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Sintaxis para pasar argumentos a rutinas escritas en C/C++, OLE o COBOL" en la página 97

Ejemplos relacionados:

- "spserver.c -- Definition of various types of stored procedures"
- "udfcli.c -- How to work with different types of user-defined functions (UDFs)"
- "spserver.sqC -- Definition of various types of stored procedures (C++)"
- "udfemsrv.sqC -- Call a variety of types of embedded SQL user-defined functions. (C++)"
- "udfemsrv.sqc -- Call a variety of types of embedded SQL user-defined functions. (C)"

Archivo de inclusión para rutinas C/C++ (sqludf.h)

El archivo de inclusión `sqludf.h` contiene estructuras, definiciones y valores que son de utilidad al escribir las rutinas. Aunque el nombre de este archivo incluye 'udf', (por razones históricas) también sirve para los procedimientos almacenados y los métodos. Cuando se compile una rutina, se tendrá que hacer referencia al directorio que contiene este archivo. Este directorio es `sqllib/include`.

El archivo de inclusión `sqludf.h` se autodescribe. A continuación mostramos un breve resumen de su contenido:

1. Definiciones de estructuras para los argumentos pasados que son estructuras:
   - Argumentos y resultado VARCHAR FOR BIT DATA
   - Argumentos y resultado LONG VARCHAR (con o sin FOR BIT DATA)
   - Argumentos y resultado LONG VARGRAPHIC
   - Argumentos y resultado SQL de todos los tipos de LOB
   - El área reutilizable
   - La estructura dbinfo
2. Las definiciones del tipo de lenguaje C para todos los tipos de datos SQL, para utilizarlos en la definición de argumentos de rutinas correspondientes a argumentos y resultado de SQL que tienen los tipos de datos. Se trata de las definiciones que tienen por nombre SQLUDF_x y SQLUDF_x_FBD, donde x es un nombre de tipo de datos de SQL y FBD representa FOR BIT DATA.

   También se incluye un tipo de lenguaje C para un argumento o resultado definido con la cláusula AS LOCATOR. Esto sólo es aplicable a las UDF y los métodos.
3. Definición de tipos de lenguaje C para argumentos del *área-reutilizable* y del *tipo-llamada*, con una definición de tipo `enum` del argumento *tipo-llamada*.
4. Macros para definir los argumentos estándar de *cola*, tanto con como sin la inclusión de argumentos de *área-reutilizable* y *tipo-llamada*. Esto corresponde a la presencia y ausencia de las palabras clave SCRATCHPAD y FINAL CALL en la definición de la función. Son los argumentos de invocación de UDF *estado-SQL*, *nombre-función*, *nombre-específico*, *mensaje-diagnóstico*, *área-reutilizable* y *tipo-llamada*. También se incluyen las definiciones para hacer referencia a estas construcciones y los diversos valores de SQLSTATE válidos.
5. Macros para comprobar si los argumentos de SQL son nulos.

Existe un archivo de inclusión correspondiente para COBOL: `sqludf.cbl`. Este archivo sólo incluye definiciones para las estructuras de área reutilizable y dbinfo.

Conceptos relacionados:

- "Manejo de tipos de datos de SQL en rutinas C/C++" en la página 171
- "Rutinas C/C++" en la página 164

Información relacionada:

- "Sintaxis para pasar argumentos a rutinas escritas en C/C++, OLE o COBOL" en la página 97
- "Tipos de datos de SQL soportados en C/C++" en la página 168

Tipos de datos de SQL soportados en C/C++

La tabla siguiente lista las correlaciones soportadas entre los tipos de datos de SQL y los tipos de datos de C para las rutinas. Junto a cada tipo de datos de C/C++, se encuentra el tipo definido correspondiente de sqludf.h.

Tabla 25. 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	sqlint16 SQLUDF_SMALLINT	Entero con signo de 16 bits
INTEGER	sqlint32 SQLUDF_INTEGER	Entero con signo de 32 bits
BIGINT	sqlint64 SQLUDF_BIGINT	Entero con signo de 64 bits
REAL FLOAT(*n*) donde 1<=n<=24	float SQLUDF_REAL	Coma flotante de precisión simple
DOUBLE FLOAT FLOAT(*n*) donde 25<=n<=53	double SQLUDF_DOUBLE	Coma flotante de precisión doble
DECIMAL(*p*, *s*)	No soportado.	Para pasar un valor decimal, defina el parámetro como de tipo de datos moldeable DECIMAL (por ejemplo, CHAR o DOUBLE) y moldee explícitamente el argumento para este tipo.
CHAR(*n*)	char[*n+1*] donde n es suficientemente grande para contener los datos 1<=*n*<=254 SQLUDF_CHAR	Serie de caracteres de longitud fija terminada en nulo
CHAR(*n*) FOR BIT DATA	char[*n+1*] donde n es suficientemente grande para contener los datos 1<=*n*<=254 SQLUDF_CHAR	Serie de caracteres de longitud fija terminada en nulo
VARCHAR(*n*)	char[*n+1*] donde n es suficientemente grande para contener los datos 1<=*n*<=32 672 SQLUDF_VARCHAR	Serie de longitud variable terminada en nulo

VARCHAR(*n*) FOR BIT DATA	struct {
	sqluint16 length;
	char[*n*]
	}
	
	1<=*n*<=32 672
	SQLUDF_VARCHAR_FBD
		Serie de caracteres de longitud variable no terminada en nulo

LONG VARCHAR	struct {
	sqluint16 length;
	char[*n*]
	}
	
	1<=*n*<=32 700
	
	SQLUDF_LONG
		Serie de caracteres de longitud variable no terminada en nulo

CLOB(*n*)	struct { sqluint32 length; char data[*n*];} 1<=*n*<=2 147 483 647 SQLUDF_CLOB	Serie de caracteres de longitud variable no terminada en nulo con indicador de longitud de serie de 4 bytes

Tabla 25. 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

BLOB(*n*)	struct {
	sqluint32 length;
	char data[n];
	}
	
	1<=*n*<=2 147 483 647
	
	SQLUDF_BLOB
		Serie binaria de longitud variable no terminada en nulo con indicador de longitud de serie de 4 bytes

DATE	char[11] SQLUDF_DATE	Serie de caracteres terminada en nulo con el formato siguiente: `aaaa-mm-dd`
TIME	char[9] SQLUDF_TIME	Serie de caracteres terminada en nulo con el formato siguiente: `hh.mm.ss`
TIMESTAMP	char[27] SQLUDF_STAMP	Serie de caracteres terminada en nulo con el formato siguiente: `aaaa-mm-dd-hh.mm.ss.nnnnnn`
LOB LOCATOR	sqluint32 SQLUDF_LOCATOR	Entero con signo de 32 bits

DATALINK	struct {
	sqluint32 version;
	char linktype[4];
	sqluint32 url_length;
	sqluint32 comment_length;
	char reserve2[8];
	char url_plus_comment[230];
	}
	
	SQLUDF_DATALINK	

Nota: Los tipos de datos siguientes sólo están disponibles en los entornos DBCS o EUC si se precompilan con la opción WCHARTYPE NOCONVERT.		
GRAPHIC(*n*)	sqldbchar[*n+1*] donde n es suficientemente grande para contener los datos 1<=*n*<=127 SQLUDF_GRAPH	Serie de caracteres de doble byte y longitud fija terminada en nulo
VARGRAPHIC(*n*)	sqldbchar[*n+1*] donde n es suficientemente grande para contener los datos 1<=*n*<=16 336 SQLUDF_GRAPH	Serie de caracteres de doble byte y longitud variable no terminada en nulo

LONG VARGRAPHIC	struct {
	sqluint16 length;
	sqldbchar[*n*]
	}
	
	1<=*n*<=16 350
	
	SQLUDF_LONGVARG
		Serie de caracteres de doble byte y longitud variable no terminada en nulo

Tabla 25. 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*)	struct {
	sqluint32 length;
	sqldbchar data[n];
	}
	
	1<=*n*<=1 073 741 823
	
	SQLUDF_DBCLOB
		Serie de caracteres de longitud variable no terminada en nulo con indicador de longitud de serie de 4 bytes

Conceptos relacionados:

- "Archivo de inclusión para rutinas C/C++ (sqludf.h)" en la página 168
- "Manejo de tipos de datos de SQL en rutinas C/C++" en la página 171
- "Rutinas C/C++" en la página 164

Manejo de tipos de datos de SQL en rutinas C/C++

En este apartado se identifican los tipos válidos para parámetros y resultados de rutinas y se especifica cómo se debe definir el argumento correspondiente en la rutina, en los lenguajes C o C++. Todos los argumentos de la rutina se deben pasar como punteros al tipo de datos adecuado. Observe que, si utiliza el archivo de inclusión `sqludf.h` y los tipos definidos en éste, puede generar automáticamente estructuras y variables del lenguaje que sean correctas para los distintos tipos de datos y compiladores. Por ejemplo, para BIGINT puede utilizar el tipo de datos SQLUDF_BIGINT a fin de ocultar las diferencias en el tipo requerido para la representación de BIGINT entre distintos compiladores.

El que gobierna el formato de los valores de argumento es el tipo de datos para cada parámetro definido en la sentencia CREATE de la rutina. A fin de obtener el valor en el formato adecuado, puede ser necesario realizar promociones a partir del tipo de datos del argumento. DB2® lleva a cabo automáticamente estas promociones sobre los valores de los argumentos. No obstante, si se especifican tipos de datos incorrectos en el código de la rutina, se producirá un comportamiento imprevisible, como, por ejemplo, pérdida de datos o terminaciones anómalas.

Para el resultado de un método o de una función escalar, es el tipo de datos especificado en la cláusula CAST FROM de la sentencia CREATE FUNCTION el que define el formato. Si no hay ninguna cláusula CAST FROM presente, define el formato el tipo de datos especificado en la cláusula RETURNS.

En el ejemplo siguiente, la presencia de la cláusula CAST FROM significa que el cuerpo de la rutina devuelve un SMALLINT y que DB2 convierte el valor a INTEGER antes de pasarlo a la sentencia en que se produce la referencia a la función:

```
... RETURNS INTEGER CAST FROM SMALLINT ...
```

En este caso, la rutina se debe escribir de forma que genere un SMALLINT, tal como se define más adelante en este apartado. Tenga en cuenta que el tipo de datos CAST FROM se debe *poder convertir* al tipo de datos de RETURNS, por lo que no es posible elegir arbitrariamente otro tipo de datos.

A continuación se muestra una lista de los tipos de SQL y sus representaciones en los lenguajes C/C++. Esta lista incluye información sobre si cada tipo es válido

como parámetro o como resultado. También se incluyen ejemplos de cómo pueden aparecer los tipos como definición de argumentos en la rutina, en los lenguajes C o C++:

- SMALLINT

  Válidos. Se representan en C como `SQLUDF_SMALLINT` o `sqlint16`.

  Ejemplo:

  ```
  sqlint16    *arg1;          /* ejemplo de SMALLINT */
  ```

  Cuando defina parámetros enteros de una rutina, considere la posibilidad de utilizar INTEGER en lugar de SMALLINT, ya que DB2 no promociona los argumentos INTEGER a SMALLINT. Por ejemplo, suponga que define una UDF del modo siguiente:

  ```
  CREATE FUNCTION SIMPLE(SMALLINT)...
  ```

  Si invoca la función SIMPLE utilizando datos INTEGER (`... SIMPLE(1)...`), recibirá un error SQLCODE -440 (SQLSTATE 42884) indicando que no se ha encontrado la función, y es posible que los usuarios finales de esta función no perciban la razón del mensaje. En el ejemplo anterior, 1 es un INTEGER, por lo que puede convertirlo a SMALLINT o puede definir el parámetro como INTEGER.

- INTEGER o INT

  Válidos. Se representan en C como `SQLUDF_INTEGER` o `sqlint32`. Debe ejecutar `#include sqludf.h` o `#include sqlsystm.h` para tomar esta definición.

  Ejemplo:

  ```
  sqlint32 *arg2;          /* ejemplo de INTEGER */
  ```

- BIGINT

  Válidos. Se representan en C como `SQLUDF_BIGINT` o `sqlint64`.

  Ejemplo:

  ```
  sqlint64 *arg3;          /* ejemplo de INTEGER */
  ```

  DB2 define el tipo `sqlint64` del lenguaje C para superar las diferencias entre definiciones del entero con signo de 64 bits en compiladores y sistemas operativos. Debe ejecutar `#include sqludf.h` o `#include sqlsystm.h` para tomar la definición.

- REAL o FLOAT(*n*) donde 1 <= *n* <= 24

  Válidos. Se representan en C como `SQLUDF_REAL` o `float`.

  Ejemplo:

  ```
  float *result;          /* ejemplo de REAL */
  ```

- DOUBLE o DOUBLE PRECISION o FLOAT o FLOAT(*n*) donde 25 <= *n* <= 53

  Válidos. Se representan en C como `SQLUDF_DOUBLE` o `double`.

  Ejemplo:

  ```
  double  *result;        /* ejemplo de DOUBLE   */
  ```

- DECIMAL(p,s) o NUMERIC(p,s)

  No válidos, puesto que no existe ninguna representación en lenguaje C. Si desea pasar un valor decimal, debe definir el parámetro como de tipo de datos DECIMAL que se puede convertir (por ejemplo, CHAR o DOUBLE) y convertir explícitamente el argumento a este tipo. En el caso de DOUBLE, no es necesario que convierta explícitamente un argumento decimal a un parámetro DOUBLE, ya que DB2 lo promociona automáticamente.

  Ejemplo:

Suponga que tiene dos columnas, WAGE como DECIMAL(5,2) y HOURS como DECIMAL(4,1), y que desea escribir una UDF para calcular el pago semanal en base al salario, el número de horas trabajadas y algunos otros factores. La UDF podría ser como la siguiente:

```
CREATE FUNCTION WEEKLY_PAY (DOUBLE, DOUBLE, ...)
       RETURNS DECIMAL(7,2) CAST FROM DOUBLE
       ...;
```

Para la UDF anterior, los dos primeros parámetros corresponden al salario y al número de horas. Invoque a la UDF WEEKLY_PAY en la sentencia de selección de SQL, del modo siguiente:

```
SELECT WEEKLY_PAY (WAGE, HOURS, ...) ...;
```

Observe que no se requiere ninguna conversión explícita porque los argumentos DECIMAL se pueden convertir a DOUBLE.

Alternativamente, puede definir WEEKLY_PAY con argumentos CHAR, del modo siguiente:

```
CREATE FUNCTION WEEKLY_PAY (VARCHAR(6), VARCHAR(5), ...)
       RETURNS DECIMAL (7,2) CAST FROM VARCHAR(10)
       ...;
```

La puede invocar así:

```
SELECT WEEKLY_PAY (CHAR(WAGE), CHAR(HOURS), ...) ...;
```

Observe que se requiere una conversión explícita porque los argumentos DECIMAL no se pueden promocionar a VARCHAR.

Una ventaja de utilizar parámetros de coma flotante es que resulta fácil realizar operaciones aritméticas sobre los valores de la rutina; una ventaja de utilizar parámetros de tipo carácter es que siempre es posible representar exactamente el valor decimal. Esto no es siempre posible con la coma flotante.

- CHAR(n) o CHARACTER(n) con o sin el modificador FOR BIT DATA.

  Válidos. Se representan en C como `SQLUDF_CHAR` o `char...[n+1]` (ésta es una serie C terminada en nulo).

  Ejemplo:

  ```
  char     arg1[14];      /* ejemplo de CHAR(13)   */
  char    *arg1;          /* también se puede aceptar */
  ```

  Para un parámetro CHAR(n), DB2 mueve *n* bytes de datos al almacenamiento intermedio y establece el byte en la posición *n+1*> para el terminador nulo (X'00'). Para un valor de RETURNS CHAR(n) o un parámetro de salida de un procedimiento almacenado que no se ha especificado como FOR BIT DATA, DB2 busca un terminador nulo dentro de los *n* primeros bytes del valor CHAR. Si se encuentra un terminador nulo, DB2 rellena los bytes restantes, hasta el byte *n*, con espacios en blanco ascii. Para un valor de RETURNS CHAR(n) o un parámetro de salida de un procedimiento almacenado especificado como FOR BIT DATA, DB2 copia encima de los *n* primeros bytes independientemente de las apariciones de terminadores nulos de serie en los *n* bytes. Los terminadores nulos de serie se tratan como datos normales.

  Tenga precaución al utilizar las funciones normales de manejo de series C en una rutina que manipule un valor FOR BIT DATA, ya que muchas de estas funciones buscan un terminador nulo para delimitar un argumento de serie y los terminadores nulos (X'00') pueden aparecer legítimamente en medio de un valor FOR BIT DATA. El uso de las funciones C en los valores FOR BIT DATA puede causar el truncamiento no deseado del valor de datos.

  Cuando defina parámetros de tipo carácter de una rutina, piense en la posibilidad de utilizar VARCHAR en lugar de CHAR, puesto que DB2 no

promociona los argumentos VARCHAR a CHAR y los literales de serie se consideran automáticamente VARCHAR. Por ejemplo, suponga que define una UDF del modo siguiente:

```
CREATE FUNCTION SIMPLE(INT,CHAR(1))...
```

Si invoca la función SIMPLE utilizando datos VARCHAR (`... SIMPLE(1,'A')...`), recibirá un error SQLCODE -440 (SQLSTATE 42884) indicando que no se ha encontrado la función, y es posible que los usuarios finales de esta función no perciban la razón del mensaje. En el ejemplo anterior, `'A'` es VARCHAR, por lo que puede convertirlo a CHAR o puede definir el parámetro como VARCHAR.

- VARCHAR(n) FOR BIT DATA o LONG VARCHAR con o sin el modificador FOR BIT DATA.

  Válidos. Representan VARCHAR(n) FOR BIT DATA en C como `SQLUDF_VARCHAR_FBD`. Representan LONG VARCHAR en C como `SQLUDF_LONG`. De lo contrario, representan estos dos tipos de SQL en C como una estructura similar a la siguiente del archivo de inclusión sqludf.h:

  ```
  struct sqludf_vc_fbd
  {
     unsigned short length;         /* longitud de los datos */
     char           data[1];        /* primer carácter de datos */
  };
  ```

  El [1] indica una matriz para el compilador. No significa que sólo se pase un carácter; dado que se pasa la dirección de la estructura, y no la estructura real, proporciona una manera de utilizar la lógica de matrices.

  Estos valores no se representan como series C terminadas en nulo porque el carácter de nulo podría ser admisible formando parte del valor de los datos. Se pasa explícitamente a la rutina la longitud de los parámetros utilizando la variable de estructura `length`. Para la cláusula RETURNS, la longitud que se pasa a la rutina es la longitud del almacenamiento intermedio. Lo que el cuerpo de la rutina debe devolver, utilizando la variable de estructura `length`, es la longitud real del valor de los datos.

  Ejemplo:

  ```
  struct sqludf_vc_fbd *arg1;  /* ejemplo de VARCHAR(n) FOR BIT DATA */
  struct sqludf_vc_fbd *result; /* y también de LONG VARCHAR FOR BIT DATA */
  ```

- VARCHAR(n) sin FOR BIT DATA.

  Válidos. Se representan en C como `SQLUDF_VARCHAR` o `char...[n+1]`. (Ésta es una serie C terminada en nulo.)

  Para un parámetro VARCHAR(n), DB2 colocará un nulo en la posición (k+1), donde k es la longitud de la serie en particular. Las funciones de manejo de series C son adecuadas para la manipulación de estos valores. Para un valor de RETURNS VARCHAR(n) o un parámetro de salida de un procedimiento almacenado, el cuerpo de la rutina debe delimitar el valor real con un nulo porque DB2 determinará la longitud del resultado a partir de este carácter de nulo.

  Ejemplo:

  ```
  char     arg2[51];      /* ejemplo de VARCHAR(50)   */
  char    *result;        /* también se puede aceptar */
  ```

- DATE

  Válidos. Se representan en C como `SQLUDF_DATE` o CHAR(10), es decir, como `char...[11]`. El valor de fecha siempre se pasa a la rutina con formato ISO:

  ```
  aaaa-mm-dd
  ```

  Ejemplo:

```
char     arg1[11];      /* ejemplo de DATE       */
char    *result;        /* también se puede aceptar */
```

Nota: Para los valores de retorno de DATE, TIME y TIMESTAMP, DB2 exige que los caracteres estén en el formato definido y, si no es así, DB2 puede malinterpretar el valor.

- TIME

  Válidos. Se representan en C como SQLUDF_TIME o CHAR(8), es decir, como char...[9]. El valor de hora siempre se pasa a la rutina con formato ISO:

  ```
  hh.mm.ss
  ```

  Ejemplo:

  ```
  char    *arg;           /* ejemplo de DATE          */
  char     result[9];     /* también se puede aceptar */
  ```

- TIMESTAMP

  Válidos. Se representan en C como SQLUDF_STAMP o CHAR(26), es decir, como char...[27]. El valor de indicación de la hora siempre se pasa con el formato siguiente:

  ```
  aaaa-mm-dd-hh.mm.ss.nnnnnn
  ```

  Ejemplo:

  ```
  char     arg1[27];      /* ejemplo de TIMESTAMP  */
  char    *result;        /* también se puede aceptar */
  ```

- GRAPHIC(n)

  Válidos. Se representan en C como SQLUDF_GRAPH o sqldbchar[n+1]. (Ésta es una serie gráfica terminada en nulo.) Tenga en cuenta que puede utilizar wchar_t[n+1] en los sistemas operativos en los que wchar_t está definido con 2 bytes de longitud; no obstante, es recomendable sqldbchar.

  Para un parámetro GRAPHIC(n), DB2 mueve *n* caracteres de doble byte al almacenamiento intermedio y establece los dos bytes siguientes en nulo. Los datos que se pasan de DB2 a una rutina están en formato DBCS y es de esperar que el resultado devuelto esté en formato DBCS. Este comportamiento es el mismo que si se utiliza la opción de precompilador WCHARTYPE NOCONVERT. Para un valor de RETURNS GRAPHIC(*n*) o un parámetro de salida de un procedimiento almacenado, DB2 busca un CHAR GRAPHIC nulo incorporado y, si lo encuentra, rellena el valor hasta *n* con caracteres GRAPHIC en blanco.

  Cuando defina parámetros gráficos de una rutina, considere la posibilidad de utilizar VARGRAPHIC en lugar de GRAPHIC, ya que DB2 no promociona los argumentos VARGRAPHIC a GRAPHIC. Por ejemplo, suponga que define una rutina del modo siguiente:

  ```
  CREATE FUNCTION SIMPLE(GRAPHIC)...
  ```

  Si invoca la función SIMPLE utilizando datos VARGRAPHIC (... SIMPLE('*literal_gráfico*')...), recibirá un error SQLCODE -440 (SQLSTATE 42884) indicando que no se ha encontrado la función, y es posible que los usuarios finales de esta función no comprendan la razón de este mensaje. En el ejemplo anterior, *literal_gráfico* es una serie DBCS literal que se interpreta como datos VARGRAPHIC, por lo que puede convertirla a GRAPHIC o puede definir el parámetro como VARGRAPHIC.

  Ejemplo:

  ```
  sqldbchar   arg1[14];      /* ejemplo de GRAPHIC(13)   */
  sqldbchar  *arg1;          /* también se puede aceptar */
  ```

- VARGRAPHIC(n)

Válidos. Se representan en C como SQLUDF_GRAPH o sqldbchar[n+1].(Ésta es una serie gráfica terminada en nulo.) Tenga en cuenta que puede utilizar wchar_t[n+1] en los sistemas operativos en los que wchar_t está definido con 2 bytes de longitud; no obstante, es recomendable sqldbchar.

Para un parámetro VARGRAPHIC(*n*), DB2 colocará un nulo gráfico en la posición (k+1), donde *k* es la longitud de la aparición en particular. Un nulo gráfico hace referencia a una situación en que todos los bytes del último carácter de la serie gráfica contienen ceros binarios ('\0's). Los datos que se pasan de DB2 a una rutina están en formato DBCS y es de esperar que el resultado devuelto esté en formato DBCS. Este comportamiento es el mismo que si se utiliza la opción de precompilador WCHARTYPE NOCONVERT. Para un valor de RETURNS VARGRAPHIC(*n*) o un parámetro de salida de un procedimiento almacenado, el cuerpo de la rutina debe delimitar el valor real con un nulo gráfico, porque DB2 determinará la longitud del resultado a partir de este carácter de nulo gráfico.

Ejemplo:

```
sqldbchar   args[51],    /* ejemplo de VARGRAPHIC(50) */
sqldbchar   *result,     /* también se puede aceptar  */
```

- LONG VARGRAPHIC

  Válidos. Se representan en C como SQLUDF_LONGVARG o como una estructura:

  ```
  struct sqludf_vg
  {
     unsigned short length;         /* longitud de los datos */
     sqldbchar      data[1];        /* primer carácter de datos */
  };
  ```

  Tenga en cuenta que, en la estructura anterior, puede utilizar wchar_t en lugar de sqldbchar en los sistemas operativos en los que wchar_t está definido con 2 bytes de longitud; no obstante, es recomendable el uso de sqldbchar.

  El [1] simplemente indica una matriz para el compilador. No significa que sólo se pase un carácter gráfico. Dado que se pasa la dirección de la estructura, y no la estructura real, se proporciona una manera de utilizar la lógica de matrices.

  No se representan como series gráficas terminadas en nulo. Se pasa explícitamente a la rutina la longitud de los parámetros, en caracteres de doble byte, utilizando la variable de estructura length. Los datos que se pasan de DB2 a una rutina están en formato DBCS y es de esperar que el resultado devuelto esté en formato DBCS. Este comportamiento es el mismo que si se utiliza la opción de precompilador WCHARTYPE NOCONVERT. Para la cláusula RETURNS o un parámetro de salida de un procedimiento almacenado, la longitud que se pasa a la rutina es la longitud del almacenamiento intermedio. Lo que el cuerpo de la rutina debe devolver, utilizando la variable de estructura length, es la longitud real del valor de los datos, en caracteres de doble byte.

  Ejemplo:

  ```
  struct sqludf_vg *arg1;  /* ejemplo de VARGRAPHIC(n)   */
  struct sqludf_vg *result; /* y también de LONG VARGRAPHIC   */
  ```

- BLOB(n) y CLOB(n)

  Válidos. Se representan en C como SQLUDF_BLOB, como SQLUDF_CLOB o como una estructura:

  ```
  struct sqludf_lob
  {
      sqluint32     length;      /* longitud en bytes */
      char          data[1];      /* primer byte del lob */
  };
  ```

El [1] simplemente indica una matriz para el compilador. No significa que sólo se pase un carácter; dado que se pasa la dirección de la estructura, y no la estructura real, proporciona una manera de utilizar la lógica de matrices.

No se representan como series C terminadas en nulo. Se pasa explícitamente a la rutina la longitud de los parámetros utilizando la variable de estructura `length`. Para la cláusula RETURNS o un parámetro de salida de un procedimiento almacenado, la longitud que se devuelve a la rutina es la longitud del almacenamiento intermedio. Lo que el cuerpo de la rutina debe devolver, utilizando la variable de estructura `length`, es la longitud real del valor de los datos.

Ejemplo:

```
struct sqludf_lob *arg1;  /* ejemplo de BLOB(n), CLOB(n) */
struct sqludf_lob *result;
```

- DBCLOB(n)

  Válidos. Se representan en C como `SQLUDF_DBCLOB` o como una estructura:

  ```
  struct sqludf_lob
  {
      sqluint32 length;      /* longitud en caracteres gráficos */
      sqldbchar data[1];         /* primer byte del lob */
  };
  ```

  Tenga en cuenta que, en la estructura anterior, puede utilizar `wchar_t` en lugar de `sqldbchar` en los sistemas operativos en los que `wchar_t` está definido con 2 bytes de longitud; no obstante, es recomendable el uso de `sqldbchar`.

  El [1] simplemente indica una matriz para el compilador. No significa que sólo se pase un carácter gráfico; dado que se pasa la dirección de la estructura, y no la estructura real, proporciona una manera de utilizar la lógica de matrices.

  No se representan como series gráficas terminadas en nulo. Se pasa explícitamente a la rutina la longitud de los parámetros utilizando la variable de estructura `length`. Los datos que se pasan de DB2 a una rutina están en formato DBCS y es de esperar que el resultado devuelto esté en formato DBCS. Este comportamiento es el mismo que si se utiliza la opción de precompilador WCHARTYPE NOCONVERT. Para la cláusula RETURNS o un parámetro de salida de un procedimiento almacenado, la longitud que se pasa a la rutina es la longitud del almacenamiento intermedio. Lo que el cuerpo de la rutina debe devolver, utilizando la variable de estructura `length`, es la longitud real del valor de los datos, con todas estas longitudes expresadas en caracteres de doble byte.

  Ejemplo:

  ```
  struct sqludf_lob *arg1;  /* ejemplo de DBCLOB(n) */
  struct sqludf_lob *result;
  ```

- Tipos diferenciados

  Válidos o no válidos en función del tipo base. Los tipos diferenciados se pasarán a la UDF con el formato del tipo base del UDT, por lo que se pueden especificar si y sólo si es válido el tipo base.

  Ejemplo:

  ```
  struct sqludf_lob *arg1; /* para tipos diferenciados basados en BLOB(n) */
  double            *arg2; /* para tipos diferenciados basados en DOUBLE  */
  char              res[5];/* para tipos diferenciados basados en CHAR(4) */
  ```

- Tipos diferenciados AS LOCATOR, o cualquier tipo de LOB AS LOCATOR

  Válidos para los parámetros y resultados de UDF y métodos. Sólo se puede utilizar para modificar tipos de LOB o cualquier tipo diferenciado que esté basado en un tipo de LOB. Su representación en C es como `SQLUDF_LOCATOR` o como entero de cuatro bytes.

El valor de localizador se puede asignar a cualquier variable del lenguaje principal de localizador que sea de un tipo compatible, y luego se puede utilizar en una sentencia de SQL. Esto significa que las variables de localizador únicamente son de utilidad en las UDF y los métodos definidos con un indicador de acceso a SQL que sea CONTAINS SQL o superior. Para que sean compatibles con las UDF y los métodos existentes, las API de localizador se siguen soportando para las UDF NOT FENCED NO SQL. No se recomienda el uso de estas API para las funciones nuevas.

Ejemplo:

```
     sqludf_locator       *arg1;  /* argumento de localizador */
     sqludf_locator       *result; /* resultado del localizador */

EXEC SQL BEGIN DECLARE SECTION;
   SQL TYPE IS CLOB LOCATOR arg_loc;
   SQL TYPE IS CLOB LOCATOR res_loc;
EXEC SQL END DECLARE SECTION;

/* Extraer algunos caracteres del medio */
/* del argumento y devolverlos         */
*arg_loc = arg1;
EXEC SQL VALUES SUBSTR(arg_loc, 10, 20) INTO :res_loc;
*result = res_loc;
```

- Tipos estructurados

  Válidos para los parámetros y resultados de UDF y métodos en que exista una función de transformación apropiada. Los parámetros de tipos estructurados se pasarán a la función o al método en el tipo de resultado de la función de transformación FROM SQL. Los resultados de tipos estructurados se pasarán en el tipo de parámetro de la función de transformación TO SQL.

- DATALINK

  Válidos. Se representan en C como `SQLUDF_DATALINK` o como una estructura similar a la siguiente del archivo de inclusión sqludf.h:

```
     struct sqludf_datalink {
       sqluint32 version;
       char      linktype[4];
       sqluint32 url_length;
       sqluint32 comment_length;
       char      reserve2[8];
       char      url_plus_comment[230];
     }
```

Conceptos relacionados:

- "Funciones de transformación y grupos de transformación" en la página 306
- "Variables gráficas del lenguaje principal en rutinas C/C++" en la página 179
- "Archivo de inclusión para rutinas C/C++ (sqludf.h)" en la página 168
- "Rutinas C/C++" en la página 164

Información relacionada:

- "Sentencia CREATE FUNCTION" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE PROCEDURE" en la publicación *Consulta de SQL, Volumen 2*
- "Tipos de datos SQL soportados en C y C++" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente*
- "Tipos de datos de SQL soportados en C/C++" en la página 168

Variables gráficas del lenguaje principal en rutinas C/C++

Generalmente, cualquier rutina escrita en C o C++ que reciba o devuelva datos gráficos a través de su entrada o salida de parámetros se debe precompilar con la opción WCHARTYPE NOCONVERT. Esto es debido a que los datos gráficos que se pasan mediante dichos parámetros se considera que están en formato DBCS, en lugar del formato de código de procesos `wchar_t`. La utilización de NOCONVERT significa que los datos gráficos manipulados en sentencias de SQL de la rutina también estarán en formato DBCS, coincidiendo así con el formato de los datos de parámetros.

Con WCHARTYPE NOCONVERT, no se produce ninguna conversión del código de caracteres entre la variable gráfica del lenguaje principal y el gestor de bases de datos. Los datos de una variable gráfica del lenguaje principal se envían al gestor de bases de datos y se reciben del mismo como caracteres DBCS inalterados. Si no utiliza WCHARTYPE NOCONVERT, también podrá manipular datos gráficos en formato `wchar_t` en una rutina; sin embargo, deberá realizar manualmente las conversiones de entrada y salida.

Se puede utilizar CONVERT en las rutinas FENCED, lo que afectará a los datos gráficos de las sentencias de SQL incluidas en la rutina, pero no a los datos pasados a través de los parámetros de la rutina. Las rutinas NOT FENCED se deben crear utilizando la opción NOCONVERT.

En resumen, los datos gráficos pasados a una rutina o devueltos por ésta mediante sus parámetros de entrada o salida están en formato DBCS, independientemente de cómo se haya precompilado con la opción WCHARTYPE.

Conceptos relacionados:

- "Opción del precompilador WCHARTYPE en C y C++" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente*
- "Opción de precompilación WCHARTYPE CONVERT" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Información relacionada:

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

Decoración de tipos de C++

Los nombres de las funciones C++ se pueden sobrecargar. Pueden coexistir dos funciones C++ con el mismo nombre si tienen argumentos distintos, por ejemplo:

```
int func( int i )
```

e

```
int func( char c )
```

Por omisión, los compiladores C++ decoran con tipos o 'despedazan' los nombres de función. Esto significa que se añaden nombres de tipo de argumento a sus nombres de función correspondientes para resolverlos, como por ejemplo en `func__Fi` y `func__Fc` para los dos ejemplos anteriores. Los nombres despedazados serán distintos en cada sistema operativo, por lo que no se puede transportar el código que utiliza explícitamente un nombre despedazado.

En los sistemas operativos Windows®, el nombre de función decorado con tipos se puede determinar a partir del archivo `.obj` (objeto).

Con el compilador Microsoft® Visual C++ en Windows, puede utilizar el mandato `dumpbin` para determinar el nombre de función decorado con tipos a partir del archivo `.obj` (objeto), del modo siguiente:

```
dumpbin /symbols myprog.obj
```

donde `myprog.obj` es el archivo de objeto del programa.

En los sistemas operativos UNIX®, el nombre de función decorado con tipos se puede determinar a partir del archivo `.o` (objeto), o de la biblioteca compartida, utilizando el mandato `nm`. Este mandato puede producir una salida considerable, por lo que se le aconseja que conduzca la salida a través de `grep` para buscar la línea correcta, del modo siguiente:

```
nm myprog.o | grep myfunc
```

donde `myprog.o` es el archivo de objeto del programa y `myfunc` es la función en el archivo fuente del programa.

La salida producida por todos estos mandatos incluye una línea con el nombre de función despedazado. Por ejemplo, en UNIX, dicha línea es parecida a la siguiente:

```
myfunc__FPlT1PsT3PcN35|     3792|unamex|          | ...
```

Una vez obtenido el nombre de función despedazado de uno de los mandatos anteriores, lo podrá utilizar en el mandato apropiado. Esto se muestra más adelante en este apartado al utilizar el nombre de función despedazado que se ha obtenido en el ejemplo de UNIX anterior. Un nombre de función despedazado obtenido en Windows se utilizaría de la misma manera.

Cuando se registra una rutina con la sentencia CREATE, la cláusula EXTERNAL NAME debe especificar el nombre de función despedazado. Por ejemplo:

```
CREATE FUNCTION myfunco(...) RETURNS...
       ...
       EXTERNAL NAME '/whatever/path/myprog!myfunc__FPlT1PsT3PcN35'
       ...
```

Si la biblioteca de rutinas no contiene nombres de función C++ sobrecargados, existe la opción de utilizar `extern "C"` para hacer que el compilador no decore con tipos los nombres de función. (Tenga en cuenta que siempre puede sobrecargar los nombres de función de SQL asignados a las UDF, ya que DB2® resuelve a qué función de biblioteca debe invocar basándose en el nombre y los parámetros que toma.)

```
#include <string.h>
#include <stdlib.h>
#include "sqludf.h"

/*------------------------------------------------------------------------*/
/* función fold: salida = la serie de entrada se dobla en el punto     */
/*                        indicado por el segundo argumento.           */
/*         entradas: CLOB,               serie de entrada              */
/*                   LONG                posición por la que doblar    */
/*         salida:   CLOB                serie doblada                 */
/*------------------------------------------------------------------------*/
extern "C" void fold(
      SQLUDF_CLOB       *in1,                   /* entrada CLOB a doblar */
   ...
   ...
}
/* fin de UDF: fold */

/*------------------------------------------------------------------------*/
/* función find_vowel:                                                    */
/*             devuelve la posición de la primera vocal                   */
/*             devuelve un error si no hay ninguna vocal                  */
/*             definida como NOT NULL CALL                                */
/*         entradas: VARCHAR(500)                                         */
/*         salida:   INTEGER                                              */
/*------------------------------------------------------------------------*/
extern "C" void findvwl(
      SQLUDF_VARCHAR    *in,                    /* entrada smallint */
   ...
   ...
}
/* fin de UDF: findvwl */
```

En este ejemplo, el compilador no decora con tipos las UDF `fold` y `findvwl`, que se deben registrar en la sentencia CREATE FUNCTION utilizando sus nombres llanos. De forma similar, si un método o procedimiento almacenado C++ se codifica con `extern "C"`, se utilizará su nombre de función no decorado en la sentencia CREATE.

Conceptos relacionados:

- “Estilos de parámetros para rutinas externas” en la página 95
- “Rutinas C/C++” en la página 164
- “Manejo de los parámetros en los procedimientos de PROGRAM TYPE MAIN o PROGRAM TYPE SUB” en la página 56

Rutinas Java

Los apartados siguientes describen cómo escribir rutinas Java.

Rutinas Java

Cuando desarrolle rutinas en Java™, es muy recomendable que las registre utilizando la cláusula PARAMETER STYLE JAVA en la sentencia CREATE. Con PARAMETER STYLE JAVA, una rutina utilizará un convenio de pase de parámetros que cumple con la especificación del lenguaje Java y las rutinas de SQLJ.

Existen algunas características de los métodos y las UDF que no se pueden implantar con PARAMETER STYLE JAVA. Son las siguientes:

- funciones de tabla

- áreas reutilizables
- acceso a la estructura DBINFO
- la posibilidad de efectuar una FINAL CALL (y una primera llamada separada) a la función o al método

Si tiene necesidad de implantar las características anteriores en una UDF o en un método, puede escribir su rutina en C, o escribirla en Java, utilizando PARAMETER STYLE DB2GENERAL. Aparte de estos casos específicos, todas las menciones que se hacen en esta documentación de las rutinas en Java supondrán que se utiliza PARAMETER STYLE JAVA.

UDF y métodos en Java:

La signatura de las UDF y los métodos con PARAMETER STYLE JAVA sigue el formato siguiente:

```
public static tipo-retorno nombre-método ( argumentos-SQL ) throws SQLException
```

tipo-retorno
: Tipo de datos del valor que la rutina escalar debe devolver. En la rutina, el valor de retorno se devuelve al invocador a través de una sentencia de retorno.

nombre-método
: Nombre del método. Durante el registro de la rutina, se especifica este valor junto con el nombre de clase en la cláusula EXTERNAL NAME de la sentencia CREATE de la rutina.

argumentos-SQL
: Corresponde a la lista de parámetros de entrada de la sentencia CREATE de la rutina.

A continuación se muestra un ejemplo de una UDF en Java que devuelve el producto de sus dos argumentos de entrada:

```
public static double product( double in1, double in2 ) throws SQLException
{
     return in1 * in2;
}
```

La sentencia CREATE FUNCTION correspondiente a esta UDF es la siguiente:

```
CREATE FUNCTION product( DOUBLE in1, DOUBLE in2 )
RETURNS DOUBLE
  LANGUAGE java
  PARAMETER STYLE java
  NO SQL
  FENCED THREADSAFE
  DETERMINISTIC
  RETURNS NULL ON NULL INPUT
  NO EXTERNAL ACTION
  EXTERNAL NAME 'myjar:udfclass.product'
```

La sentencia anterior supone que el método se halla en una clase denominada `udfclass` que reside en un archivo JAR catalogado en la base de datos con el ID de Jar `myjar`.

Procedimientos almacenados en Java:

La signatura de los procedimientos almacenados con PARAMETER STYLE JAVA sigue este formato:

```
public static void nombre-método ( argumentos-SQL,
ResultSet[] matriz-conjuntos-resultados )
                  throws SQLException
```

nombre-método
: Nombre del método. Durante el registro de la rutina, se especifica este valor junto con el nombre de clase en la cláusula EXTERNAL NAME de la sentencia CREATE PROCEDURE.

argumentos-SQL
: Corresponde a la lista de parámetros de entrada de la sentencia CREATE PROCEDURE. Los parámetros de modalidad OUT o INOUT se pasan como matrices de un solo elemento. Para cada conjunto de resultados que se especifique en la cláusula DYNAMIC RESULT SETS de la sentencia CREATE PROCEDURE, se añade a la lista de parámetros una matriz de un solo elemento del tipo ResultSet.

matriz-conjuntos-resultados
: Nombre de la matriz de objetos ResultSet. Por cada conjunto de resultados declarado en el parámetro DYNAMIC RESULT SETS de la sentencia CREATE PROCEDURE, se debe declarar un parámetro del tipo ResultSet[] en la signatura del método Java.

A continuación, se muestra un ejemplo de un procedimiento almacenado escrito en Java que acepta un parámetro de entrada y luego devuelve un parámetro de salida y un conjunto de resultados:

```
public static void javastp( int inparm, int[] outparm, ResultSet[] rs )
                  throws SQLException
{
  Connection con = DriverManager.getConnection( "jdbc:default:connection");
  PreparedStatement stmt = null;
  String sql = SELECT value FROM table01 WHERE index = ?";

//Preparar la consulta con el valor del índice
  stmt = con.prepareStatement( sql );
  stmt.setInt( 1, inparm );

//Ejecutar la consulta y establecer el parámetro de salida
  rs[0] = stmt.executeQuery();
  outparm[0] = inparm + 1;

//Cerrar los recursos abiertos
  if (stmt != null) stmt.close();
  if (con != null) con.close();

  return;
}
```

La sentencia CREATE PROCEDURE correspondiente a este procedimiento almacenado es la siguiente:

```
CREATE PROCEDURE javaproc( IN in1 INT, OUT out1 INT )
  LANGUAGE java
  PARAMETER STYLE java
  DYNAMIC RESULT SETS 1
  FENCED THREADSAFE
  EXTERNAL NAME 'myjar:stpclass.javastp'
```

La sentencia anterior supone que el método se halla en una clase denominada `stpclass`, que existe en un archivo JAR catalogado en la base de datos con el ID de Jar `myjar`.

Notas:

1. Las rutinas con PARAMETER STYLE JAVA utilizan excepciones para pasar los datos sobre errores al invocador. Si desea información completa, lo que incluye la pila de llamadas de excepción, consulte el registro de notificaciones de administración. Aparte de este detalle, no existe ninguna otra consideración especial para invocar las rutinas con PARAMETER STYLE JAVA.
2. En las rutinas en Java no se soportan las llamadas JNI. Sin embargo, es posible invocar funciones de C desde rutinas en Java anidando una invocación de una rutina en C. Esto implica mover la función de C deseada a una rutina, registrarla e invocarla desde dentro de la rutina en Java.

Conceptos relacionados:

- "Rutinas DB2GENERAL" en la página 357
- "Modelo de ejecución de funciones de tabla para Java" en la página 65

Tareas relacionadas:

- "Depuración de procedimientos almacenados de Java" en la página 189
- "Creación de rutinas JDBC" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de rutinas SQLJ" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Información relacionada:

- "Sentencia CREATE TYPE (Estructurado)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE METHOD" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE FUNCTION (Escalar externa)" en la publicación *Consulta de SQL, Volumen 2*
- "Tipos de datos de SQL soportados en Java" en la página 185
- "Administración de los archivos JAR en el servidor de bases de datos" en la página 187
- "Ejemplos de JDBC" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Ejemplos de SQLJ" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Sentencia CREATE PROCEDURE (Externo)" en la publicación *Consulta de SQL, Volumen 2*

Ejemplos relacionados:

- "SpServer.java -- Provide a variety of types of stored procedures to be called from (JDBC)"
- "UDFjsrv.java -- Provide UDFs to be called by UDFjcli.java (JDBC)"
- "UDFsqlsv.java -- Provide UDFs to be called by UDFsqlcl.java (JDBC)"
- "UDFsrv.java -- Provide UDFs to be called by UDFcli.java (JDBC)"
- "SpServer.sqlj -- Provide a variety of types of stored procedures to be called from (SQLj)"
- "UDFjsrv.java -- Provide UDFs to be called by UDFjcli.sqlj (SQLj)"
- "UDFsrv.java -- Provide UDFs to be called by UDFcli.sqlj (SQLj)"

Tipos de datos de SQL soportados en Java

La tabla siguiente muestra el equivalente en Java de cada tipo de datos de SQL, según la especificación JDBC correspondiente a correlaciones de tipos de datos. El controlador JDBC convierte los datos que se intercambian entre la aplicación y la base de datos utilizando el siguiente esquema de correlación. Utilice estas correlaciones en sus aplicaciones Java y sus UDF y procedimientos PARAMETER STYLE JAVA.

Nota: No existe soporte de variable del lenguaje principal para el tipo de datos DATALINK en ninguno de los lenguajes de programación soportados por DB2.

Tabla 26. Tipos de datos de SQL correlacionados con declaraciones Java

Tipo de columna de SQL	Tipo de datos de Java	Descripción del tipo de columna de SQL
SMALLINT (500 ó 501)	short, boolean	Entero con signo de 16 bits
INTEGER (496 ó 497)	int	Entero con signo de 32 bits
BIGINT [1] (492 ó 493)	long	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)	java.math.BigDecimal	Decimal empaquetado
CHAR(*n*) (452 ó 453)	java.lang.String	Serie de caracteres de longitud fija con longitud *n*, donde *n* va de 1 a 254
CHAR(*n*) FOR BIT DATA	byte[]	Serie de caracteres de longitud fija con longitud *n*, donde *n* va de 1 a 254
VARCHAR(*n*) (448 ó 449)	java.lang.String	Serie de caracteres de longitud variable
VARCHAR(*n*) FOR BIT DATA	byte[]	Serie de caracteres de longitud variable
LONG VARCHAR (456 ó 457)	java.lang.String	Serie de caracteres de longitud variable larga
LONG VARCHAR FOR BIT DATA	byte[]	Serie de caracteres de longitud variable larga
BLOB(*n*) (404 ó 405)	java.sql.Blob	Serie binaria de longitud variable y objeto grande
CLOB(*n*) (408 ó 409)	java.sql.Clob	Serie de caracteres de longitud variable y objeto grande
DBCLOB(*n*) (412 ó 413)	java.sql.Clob	Serie de caracteres de doble byte de longitud variable y objeto grande
DATE (384 ó 385)	java.sql.Date	Serie de caracteres de 10 bytes
TIME (388 ó 389)	java.sql.Time	Serie de caracteres de 8 bytes
TIMESTAMP (392 ó 393)	java.sql.Timestamp	Serie de caracteres de 26 bytes

Tabla 26. Tipos de datos de SQL correlacionados con declaraciones Java (continuación)

Tipo de columna de SQL	Tipo de datos de Java	Descripción del tipo de columna de SQL
GRAPHIC (*n*) (468 ó 469)	java.lang.String	Serie de caracteres de doble byte de longitud fija
VARGRAPHIC (*n*) (464 ó 465)	java.lang.String	Serie de caracteres de doble byte variable no terminada en nulo con indicador de longitud de serie de 2 bytes
LONGVARGRAPHIC (472 ó 473)	java.lang.String	Serie de caracteres de doble byte variable no terminada en nulo con indicador de longitud de serie de 2 bytes

Nota:

1. Respecto a las aplicaciones Java conectadas desde un cliente DB2 UDB Versión 8.1 a un servidor DB2 UDB Versión 7.1 (o 7.2), tenga en cuenta lo siguiente: cuando se utiliza el método getObject() para recuperar un valor BIGINT, se devuelve un objeto java.math.BigDecimal.

Dónde se deben colocar las clases de Java

Puede utilizar los archivos de clases de Java™ individuales para los procedimientos almacenados y las UDF o recoger los archivos de clases en archivos JAR e instalar el archivo JAR en la base de datos. Si decide utilizar archivos JAR, consulte la descripción sobre el registro de procedimientos almacenados y funciones de Java a fin de obtener más información.

Nota: Si actualiza o sustituye los archivos de clases de rutinas de Java, debe emitir una sentencia CALL SQLJ.REFRESH_CLASSES() para permitir que DB2® cargue las clases actualizadas. Si desea más información sobre la sentencia CALL SQLJ.REFRESH_CLASSES(), consulte la descripción sobre cómo actualizar las clases de Java para las rutinas.

Para permitir que DB2 encuentre y utilice las rutinas de lenguaje Java (procedimientos almacenados, UDF o métodos), debe almacenar los archivos de clases correspondientes de este modo:

Sistemas operativos Unix y Windows®
En cualquier vía de acceso de directorio especificada por la variable CLASSPATH. Es recomendable que almacene los archivos de clases de Java asociados con rutinas de DB2 en el *directorio de función*, `/u/$DB2INSTANCE/sqllib/function`, donde `/u/$DB2INSTANCE` es el directorio asociado con el gestor de bases de datos activo actualmente.

La JVM invocada por DB2 utiliza la variable de entorno CLASSPATH para localizar los archivos de Java. DB2 añade de forma automática el directorio de función y sqllib/java/db2java.zip delante del valor de CLASSPATH a fin de que no sea necesario hacerlo manualmente. Es recomendable almacenar los archivos de clases de Java asociados con rutinas de DB2 en el directorio de función. Las clases asociadas con las rutinas no protegidas se deben almacenar en el subdirectorio `/u/$DB2INSTANCE/sqllib/function/unfenced`.

Para establecer el entorno de forma que DB2 encuentre la JVM, puede establecer el parámetro de configuración *jdk_path* o utilizar el valor por omisión. Además, es posible que tenga que establecer el parámetro de configuración *java_heap_sz* para aumentar el tamaño de almacenamiento dinámico de la aplicación.

Nota: Si declara que una clase forma parte de un paquete de Java, cree, en el directorio de función, los subdirectorios que correspondan a los nombres de clase completamente calificados y coloque los archivos de clases relacionados en el subdirectorio adecuado. Por ejemplo, si crea la clase `ibm.tests.test1` para un sistema Linux, almacene el archivo de código de bytes de Java correspondiente (denominado test1.class) en sqllib/function/ibm/tests.

Tareas relacionadas:

- "Actualización de rutinas Java (procedimientos almacenados, UDF y métodos) para la ejecución" en la página 187

Información relacionada:

- "java_heap_sz - Maximum Java interpreter heap size configuration parameter" en la publicación *Administration Guide: Performance*
- "jdk_path - Software Developer's Kit for Java installation path configuration parameter" en la publicación *Administration Guide: Performance*

Actualización de rutinas Java (procedimientos almacenados, UDF y métodos) para la ejecución

Procedimiento:

Cuando actualice clases de rutinas Java, también debe emitir una sentencia CALL SQLJ.REFRESH_CLASSES() para forzar que DB2 cargue las nuevas clases. Si no emite la sentencia CALL SQLJ.REFRESH_CLASSES() después de actualizar las clases de rutinas Java, DB2 continúa utilizando las versiones anteriores de las clases. La sentencia CALL SQLJ.REFRESH_CLASSES() sólo se aplica a rutinas FENCED. DB2 renueva las clases cuando se produce una operación COMMIT o ROLLBACK.

Nota: No es posible actualizar las rutinas NOT FENCED sin detener y reiniciar el gestor de bases de datos.

Conceptos relacionados:

- "Rutinas Java" en la página 181

Información relacionada:

- "Tabla de depuración de Java DB2DBG.ROUTINE_DEBUG" en la página 193
- "Clases de Java para rutinas DB2GENERAL" en la página 362

Administración de los archivos JAR en el servidor de bases de datos

Los archivos de clases de Java que utiliza para implantar una rutina deben residir en un archivo JAR que habrá instalado en la base de datos o en la vía de acceso de clases (CLASSPATH) correcta correspondiente a su sistema operativo. El cargador de clases de DB2 busca en las clases y los archivos JAR de CLASSPATH y recogerá la primera clase que encuentre con el nombre especificado.

Para instalar, sustituir o eliminar un archivo JAR en una instancia de DB2, utilice los procedimientos almacenados que se proporcionan con DB2:

Instalación

```
sqlj.install_jar( url-jar, id-jar )
```

Nota: El ID de autorización de la sentencia del que llama a sqlj.install_jar debe tener al menos uno de los privilegios siguientes:

- Privilegio CREATEIN para el esquema especificado implícita o explícitamente
- Autorización SYSADM o DBADM

Sustitución

```
sqlj.replace_jar( url-jar, id-jar )
```

Eliminación

```
sqlj.remove_jar( id-jar )
```

- *url-jar*: El URL que contiene el archivo JAR que se debe instalar o sustituir. El único esquema de URL que recibe soporte es 'file:'.
- *id-jar*: Un identificador de serie exclusivo, con una longitud de 128 bytes como máximo. Especifica el identificador de JAR incluido en la base de datos que se asocia con el archivo de *url-jar*.

Nota: Cuando se invocan desde aplicaciones, los procedimientos almacenados sqlj.install_jar y sqlj.remove_jar tienen un parámetro adicional. Se trata de un valor de entero que indica el uso del descriptor de despliegue en el archivo JAR especificado. Actualmente, el parámetro de despliegue no está soportado y cualquier invocación que especifique un valor distinto de cero se rechazará.

A continuación, se muestran unos cuantos ejemplos sobre cómo se utilizan los procedimientos almacenados de gestión de archivos JAR citados antes.

Para registrar un JAR ubicado en la vía de acceso /home/bob/bobsjar.jar con la instancia de base de datos como MYJAR:

```
CALL sqlj.install_jar( 'file:/home/bob/bobsjar.jar', 'MYJAR' )
```

Los mandatos de SQL posteriores que utilicen el archivo `bobsjar.jar` harán referencia al mismo con el nombre `MYJAR`.

Para sustituir MYJAR por un JAR distinto que contiene algunas clases actualizadas:

```
CALL sqlj.replace_jar( 'file:/home/bob/bobsnewjar.jar', 'MYJAR' )
```

Para eliminar MYJAR de los catálogos de base de datos:

```
CALL sqlj.remove_jar( 'MYJAR' )
```

Nota: En sistemas operativos Windows, DB2 almacena los archivos JAR en la vía de acceso especificada por el valor de registro específico de la instancia *DB2INSTPROF*. Para que los archivos JAR sean exclusivos para una instancia, debe especificar para *DB2INSTPROF* un valor exclusivo que corresponda a dicha instancia.

Conceptos relacionados:

- "Dónde se deben colocar las clases de Java" en la página 186
- "Rutinas Java" en la página 181
- "Consideraciones sobre la gestión de bibliotecas y clases" en la página 30

Contextos de conexión en rutinas SQLJ

Con la introducción de rutinas de varias hebras en DB2® Universal Database, Versión 8, es importante que las rutinas de SQLJ eviten la utilización del contexto de conexión por omisión. Es decir, cada sentencia SQL debe indicar explícitamente el objeto de ConnectionContext (contexto de conexión) y dicho contexto debe replicarse de modo explícito en el método Java™. Por ejemplo, en los anteriores releases de DB2, una rutina de SQLJ podía escribirse del siguiente modo:

```
class myClass
{
  public static void myRoutine( short myInput )
  {
    DefaultContext ctx = DefaultContext.getDefaultContext();
    #sql { some SQL statement };
  }
}
```

Esta utilización del contexto por omisión hace que todas las hebras de un entorno con varias hebras utilicen el mismo contexto de conexión, el cual, a su vez, dará como resultado que se produzcan anomalías inesperadas.

La rutina SQLJ anterior debe cambiarse del siguiente modo:

```
#context MyContext;

class myClass
{
  public static void myRoutine( short myInput )
  {
    MyContext ctx = new MyContext( "jdbc:default:connection", false );
    #sql [ctx] { some SQL statement };
    ctx.close();
  }
}
```

De este modo, cada invocación de la rutina creará su propio ConnectionContext (Contexto de conexión) exclusivo (y su propia conexión JDBC subyacente), lo cual evitará interferencias inesperadas por parte de hebras simultáneas.

Conceptos relacionados:

- "Rutinas Java" en la página 181
- "Pasos básicos para escribir una aplicación SQLJ" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente*
- "Sentencias SQL en una aplicación SQLJ" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente*

Depuración de procedimientos almacenados en Java

Los apartados siguientes describen cómo depurar procedimientos almacenados Java.

Depuración de procedimientos almacenados de Java

DB2 proporciona la posibilidad de depurar de forma interactiva un procedimiento almacenado escrito en JDBC cuando se ejecuta en un servidor AIX, Linux, del Entorno operativo Solaris, Windows NT o Windows 2000. La manera más fácil de depurar procedimientos almacenados de Java es mediante el Centro de Desarrollo de DB2.

El Depurador distribuido 9.2 de DB2 debe configurarse correctamente para permitir la depuración de procedimientos almacenados de Java mientras se trabaja con DB2. El Depurador distribuido se incluye en todas las opciones de paquetes de DB2 UDB Versión 8. El Depurador distribuido está configurado para funcionar con el nivel estándar SDK 1.3.1 que se instala con DB2 UDB Versión 8. Si utiliza un nivel distinto de SDK, debe actualizar la configuración del gestor de bases de datos de DB2 con el siguiente mandato en el indicador de línea de mandatos de DB2:

```
db2 update dbm cfg using jdk_path <jdk131 vía>
```

Procedimiento:

Para depurar procedimientos almacenados en Java:

1. Prepare la depuración.
2. Llene la tabla de depuración.
3. Invoque al depurador.

Tareas relacionadas:

- "Preparación para depurar procedimientos almacenados Java" en la página 190
- "Cómo llenar la tabla de depuración" en la página 192
- "Invocación del programa de depuración" en la página 191
- "Depuración de rutinas" en la página 42

Preparación para depurar procedimientos almacenados Java

Cuando se prepare para depurar de forma interactiva un procedimiento almacenado Java, trabajará con el procedimiento almacenado, el cliente y el servidor.

Procedimiento:

Para preparar para depurar procedimientos almacenados Java:

1. Compile el procedimiento almacenado en modalidad de depuración siguiendo la documentación de SDK.
2. Prepare el servidor.

   Si el código fuente está en el servidor, establezca la variable de entorno CLASSPATH de modo que incluya el directorio del código fuente Java o almacene el código fuente en el directorio de función, como se describe en el tema Administración de los archivos JAR en el servidor de bases de datos.
3. Establezca las variables de entorno del cliente.

   Si el código fuente se almacena en el cliente, establezca la variable de entorno DB2_DBG_PATH en el directorio que contiene el código fuente del procedimiento almacenado.
4. Cree la tabla de depuración.

   Si no utiliza el Centro de Desarrollo para invocar al programa de depuración, cree la tabla de depuración con el mandato siguiente:

   ```
   db2 -tf sqllib/misc/db2debug.ddl
   ```

   Nota: En entornos de bases de datos particionadas, el grupo de particiones de bases de datos por omisión es IBMDEFAULTGROUP para el espacio de tabla USERSPACE1 y abarca todas las particiones de bases de datos. Para mejorar el rendimiento de la depuración de procedimientos almacenados en un entorno de bases de datos particionadas, debe tener una sola

partición coordinadora en la que se producirá la depuración y debe definir un grupo de particiones de base de datos que sólo contenga dicha partición de base de datos.

5. Configure el Depurador distribuido:
   a. Desde un indicador de mandatos de DOS, entre el mandato siguiente DB2SET: `db2set DB2ROUTINE_DEBUG=on`.
   b. Si trabaja con un sistema operativo UNIX, complete los pasos siguientes (los usuarios de Windows se saltan este paso y continúan en el paso c):
      1) `mkdir sqllib/function/src`
      2) `chmod 777 sqllib/function/src`
      3) `chmod 777 /home/youruserid/.DbgProf (algunas versiones posteriores del Depurador distribuido, como por ejemplo 9.2.3)`
   c. Inicie DB2 (o reinicie DB2 si ya se ha iniciado) entrando el mandato siguiente desde el indicador de mandatos: `db2start`
   d. Inicie el daemon de cliente entrando el mandato siguiente en el indicador de mandatos:

      ```
      idebug -qdaemon -quiport=número_puerto
      ```

      donde *quiport* es un número de puerto TCP/IP no utilizado. Si no se suministra un valor, el programa de depuración utilizará el número 8000 como número de puerto por omisión.

Ahora está preparado para llenar la tabla de depuración.

Conceptos relacionados:

- "Dónde se deben colocar las clases de Java" en la página 186

Tareas relacionadas:

- "Cómo llenar la tabla de depuración" en la página 192
- "Invocación del programa de depuración" en la página 191
- "Depuración de rutinas" en la página 42

Información relacionada:

- "Tabla de depuración de Java DB2DBG.ROUTINE_DEBUG" en la página 193
- "Administración de los archivos JAR en el servidor de bases de datos" en la página 187

Invocación del programa de depuración

En el programa de depuración, puede examinar el código fuente, visualizar variables y establecer puntos de interrupción en dicho código.

Procedimiento:

Después de realizar la preparación para la depuración y de llenar la tabla de depuración, llame al procedimiento almacenado que desea depurar. Esta acción invocará al programa de depuración en el cliente utilizando la dirección IP que ha especificado en la tabla de depuración.

Para iniciar la depuración de procedimientos almacenados de Java en el Centro de desarrollo, utilice el Asistente para crear un nuevo procedimiento almacenado de Java. Desde el panel de opciones, seleccione la de **habilitar la depuración**.

Para depurar procedimientos almacenados de Java existentes que no se han creado anteriormente con la opción de **habilitar la depuración**:

1. Desde la carpeta Procedimientos almacenados, pulse el botón derecho del ratón y seleccione **Crear para la depuración**.
2. Ejecute el procedimiento almacenado en modalidad de depuración seleccionando el icono **Ejecutar/Depurar** desde la barra de herramientas.

Para obtener más información sobre el funcionamiento del Depurador distribuido, consulte la ayuda en línea del producto Depurador distribuido.

Tareas relacionadas:

- "Preparación para depurar procedimientos almacenados Java" en la página 190
- "Cómo llenar la tabla de depuración" en la página 192
- "Depuración de rutinas" en la página 42

Información relacionada:

- "Tabla de depuración de Java DB2DBG.ROUTINE_DEBUG" en la página 193

Cómo llenar la tabla de depuración

La tabla de depuración contiene información sobre los procedimientos almacenados que depura y el entorno cliente/servidor en el que realiza la depuración. Sólo los DBA o los usuarios con privilegios INSERT, UPDATE o DELETE sobre la tabla pueden manipular valores directamente en la tabla base DB2DBG.ROUTINE_DEBUG. Sin embargo, a no ser que el DBA haya añadido más restricciones, cualquiera puede añadir, actualizar o suprimir filas a través de la vista de usuario, DB2DBG.ROUTINE_DEBUG_USER. En el resto de este apartado se supone que se llena la tabla a través de la vista de usuario.

Procedimiento:

Si utiliza el Centro de Desarrollo para invocar la depuración, puede utilizar el programa de depuración a fin de llenar y gestionar la tabla de depuración. De lo contrario, para habilitar el soporte de depuración para un determinado procedimiento almacenado, emita el siguiente mandato desde el CLP:

```
DB2 INSERT INTO db2dbg.routine_debug_user (AUTHID, TYPE,
   ROUTINE_SCHEMA, SPECIFICNAME, DEBUG_ON, CLIENT_IPADDR)
VALUES ('idaut', 'S', 'esquema', 'nombre_proc', 'Y', 'núm_IP')
```

donde:

idaut El nombre de usuario que se utiliza para depurar el procedimiento almacenado, es decir, el nombre de usuario utilizado para conectar con la base de datos.

esquema
El nombre de esquema correspondiente al procedimiento almacenado.

nombre_proc
El nombre específico del procedimiento almacenado. Es el nombre específico que se proporcionó en el mandato CREATE PROCEDURE, o un identificador generado por el sistema, si no se ha suministrado ningún nombre específico.

núm_IP
La dirección IP en el formato *nnn.nnn.nnn.nnn* del cliente utilizado para depurar el procedimiento almacenado.

Por ejemplo, para habilitar la depuración correspondiente al procedimiento almacenado *MySchema.myProc* por parte del usuario *USER1* con el cliente de depuración ubicado en la dirección IP 192.168.111.222, escriba el mandato siguiente:

```
DB2 INSERT INTO db2dbg.routine_debug_user (AUTHID, TYPE,
       ROUTINE_SCHEMA, SPECIFICNAME, DEBUG_ON, CLIENT_IPADDR)
    VALUES ('USER1', 'S', 'MySchema', 'myProc', 'Y', '192.168.111.222')
```

Si elimina un procedimiento almacenado, su información de depuración no se suprime automáticamente de la tabla de depuración. La información de depuración correspondiente a procedimientos almacenados no existentes no puede dañar la base de datos ni la instancia. Sin embargo, la información de depuración antigua puede causar cierta confusión en caso de volver a crear un procedimiento almacenado. Si desea mantener la tabla de depuración sincronizada con el catálogo de DB2, deberá suprimir la información de depuración de forma manual.

Ahora está preparado para invocar al programa de depuración.

Tareas relacionadas:

- "Preparación para depurar procedimientos almacenados Java" en la página 190
- "Invocación del programa de depuración" en la página 191
- "Depuración de rutinas" en la página 42

Información relacionada:

- "Tabla de depuración de Java DB2DBG.ROUTINE_DEBUG" en la página 193

Tabla de depuración de Java DB2DBG.ROUTINE_DEBUG

Tanto si se crea la tabla de depuración de forma manual como si lo hace a través del Centro de Desarrollo, la tabla de depuración se denomina DB2DBG.ROUTINE_DEBUG y tiene la siguiente definición:

Tabla 27. Definición de la tabla DB2DBG.ROUTINE_DEBUG

Nombre de columna	Tipo de datos	Atributos	Descripción
AUTHID	VARCHAR(128)	NOT NULL, DEFAULT USER	El id de autorización de la aplicación bajo el cual se va a llevar a cabo la depuración correspondiente a este procedimiento almacenado. Es el ID de usuario que se proporcionó al establecer conexión con la base de datos.
TYPE	CHAR(1)	NOT NULL	Valores válidos: 'S' (procedimiento)
ROUTINE_SCHEMA	VARCHAR(128)	NOT NULL	Nombre de esquema del procedimiento almacenado que se va a depurar.
SPECIFICNAME	VARCHAR(18)	NOT NULL	Nombre específico del procedimiento almacenado que se va a depurar.
DEBUG_ON	CHAR(1)	NOT NULL, DEFAULT 'N'	Valores válidos: • **Y** - habilita la depuración para el procedimiento almacenado. • **N** - inhabilita la depuración para el procedimiento almacenado. Éste es el valor por omisión.
CLIENT_IPADDR	VARCHAR(15)	NOT NULL	Dirección IP del cliente que realiza la depuración con el formato *nnn.nnn.nnn.nnn*

Tabla 27. Definición de la tabla DB2DBG.ROUTINE_DEBUG (continuación)

Nombre de columna	Tipo de datos	Atributos	Descripción
CLIENT_PORT	INTEGER	NOT NULL, DEFAULT 8000	Puerto de la comunicación de depuración. El valor por omisión es 8000.
DEBUG_STARTN	INTEGER	NOT NULL	No se utiliza.
DEBUG_STOPN	INTEGER	NOT NULL	No se utiliza.
La clave primaria de esta tabla es AUTHID, TYPE, ROUTINE_SCHEMA, SPECIFICNAME.			

La vista DB2DBG.ROUTINE_DEBUG_USER limita el acceso a esta tabla únicamente a las filas que pertenecen al usuario conectado con la base de datos.

Tareas relacionadas:

- "Depuración de procedimientos almacenados de Java" en la página 189
- "Preparación para depurar procedimientos almacenados Java" en la página 190
- "Cómo llenar la tabla de depuración" en la página 192
- "Invocación del programa de depuración" en la página 191
- "Depuración de rutinas" en la página 42

Rutinas de automatización de OLE

Los apartados siguientes describen cómo escribir rutinas de automatización de OLE.

Diseño de 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, sus aplicaciones, independientemente del lenguaje en el que estén escritas, podrán exponer sus propiedades y métodos en objetos de automatización de OLE. Otras aplicaciones, como Lotus® Notes o Microsoft Exchange, pueden integrar después estos objetos aprovechando estas propiedades y métodos a través de la automatización de OLE.

Las aplicaciones que exponen las propiedades y los métodos se denominan objetos o servidores de automatización de OLE, y las aplicaciones que acceden a estas propiedades y métodos se denominan controladores de automatización de OLE. Los servidores de automatización de OLE son componentes COM (objetos) que implantan la interfaz IDispatch de OLE. Un controlador de automatización de OLE es un cliente COM que se comunica con el servidor de automatización a través de su interfaz IDispatch. COM es el fundamento de OLE. Para las rutinas de automatización de OLE, DB2® actúa como controlador de automatización de OLE. Mediante este mecanismo, DB2 puede invocar métodos de objetos de automatización de OLE como rutinas externas.

Observe que en todos los temas de automatización de OLE se supone que el usuario está familiarizado con los términos y conceptos de la automatización de OLE. Para tener una visión general de la automatización de OLE, consulte la publicación *Microsoft Corporation: The Component Object Model Specification* de octubre de 1995. Para conocer detalles sobre la automatización de OLE, consulte la publicación *OLE Automation Programmer's Reference*, Microsoft Press, 1996, ISBN 1-55615-851-3.

Conceptos relacionados:

- "Consideraciones sobre instancias de objetos y el área reutilizable y las rutinas de OLE" en la página 196
- "Rutinas de automatización de OLE en BASIC y C++" en la página 198

Tareas relacionadas:

- "Creación de rutinas de automatización de OLE" en la página 195

Información relacionada:

- "Tipos de datos de SQL soportados en la automatización de OLE" en la página 197

Creación de rutinas de automatización de OLE

Las rutinas de automatización de OLE se implantan como métodos públicos de los objetos de automatización de OLE. Los objetos de automatización de OLE los tiene que poder crear externamente un controlador de automatización de OLE, en este caso DB2, y soportan un enlace tardío (también denominado enlace basado en IDispatch). Los objetos de automatización de OLE se deben registrar en el registro de Windows con un identificador de clase (CLSID) y, opcionalmente, con un ID programático de OLE (progID) para identificar al objeto de automatización. El progID puede identificar un servidor de automatización de OLE interno del proceso (.DLL) o local (.EXE), o un servidor remoto a través de DCOM (Distributed COM).

Procedimiento:

Para registrar las rutinas de automatización de OLE:

Después de codificar un objeto de automatización de OLE, tiene que crear los métodos del objeto como rutinas utilizando la sentencia CREATE. Crear rutinas de automatización de OLE es muy similar a registrar rutinas C o C++, pero se deben utilizar las opciones siguientes:

- LANGUAGE OLE
- FENCED NOT THREADSAFE, ya que las rutinas de automatización de OLE se deben ejecutar en modalidad FENCED, pero no se pueden ejecutar como THREADSAFE.

El nombre externo consiste en el progID de OLE que identifica el objeto de automatización de OLE y el nombre de método, separados por un ! (signo de exclamación):

```
CREATE FUNCTION bcounter () RETURNS INTEGER
  EXTERNAL NAME 'bert.bcounter!increment'
  LANGUAGE OLE
  FENCED
  NOT THREADSAFE
  SCRATCHPAD
  FINAL CALL
  NOT DETERMINISTIC
  NULL CALL
  PARAMETER STYLE DB2SQL
  NO SQL
  NO EXTERNAL ACTION
  DISALLOW PARALLEL;
```

Los convenios de llamada para las implantaciones de métodos de OLE son idénticos a los utilizados para las rutinas escritas en C o C++. Una implantación del método anterior en el lenguaje BASIC tiene el aspecto siguiente (tenga en cuenta que, en BASIC, los parámetros se definen por omisión como llamadas por referencia):

```
Public Sub increment(output As Long, _
                     indicator As Integer, _
                     sqlstate As String, _
                     fname As String, _
                     fspecname As String, _
                     sqlmsg As String, _
                     scratchpad() As Byte, _
                     calltype As Long)
```

Conceptos relacionados:

- "Automatización de Object Linking and Embedding (OLE) con Visual Basic" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Automatización de Object Linking and Embedding (OLE) con Visual C++" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Diseño de rutinas de automatización de OLE" en la página 194
- "Consideraciones sobre instancias de objetos y el área reutilizable y las rutinas de OLE" en la página 196
- "Rutinas de automatización de OLE en BASIC y C++" en la página 198

Información relacionada:

- "Sentencia CREATE TYPE (Estructurado)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE FUNCTION (Escalar externa)" en la publicación *Consulta de SQL, Volumen 2*
- "Ejemplos de Object Linking and Embedding (OLE)" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Sentencia CREATE PROCEDURE (Externo)" en la publicación *Consulta de SQL, Volumen 2*
- "Tipos de datos de SQL soportados en la automatización de OLE" en la página 197

Consideraciones sobre instancias de objetos y el área reutilizable y las rutinas de OLE

Los métodos y las UDF de automatización de OLE (métodos de objetos de automatización de OLE) se aplican a instancias de objetos de automatización de OLE. DB2® crea una instancia de un objeto para cada referencia a un método o a una UDF en una sentencia de SQL. Una instancia de un objeto se puede volver a utilizar para invocaciones posteriores al método de referencia del método o de la UDF en una sentencia de SQL, o se puede liberar la instancia después de la invocación al método y crear una nueva instancia para cada invocación posterior al mismo. Se puede especificar el comportamiento adecuado mediante la opción SCRATCHPAD de la sentencia CREATE. Para la cláusula LANGUAGE OLE, la opción SCRATCHPAD tiene una semántica adicional en comparación con C o C++, la cual consiste en que se crea una sola instancia de un objeto y se vuelve a utilizar para la consulta entera, mientras que, si se especifica NO SCRATCHPAD, se puede crear una nueva instancia del objeto cada vez que se invoque un método.

La utilización del área reutilizable permite que un método conserve información sobre el estado en variables de instancias del objeto, a lo largo de las invocaciones a los métodos o funciones. También se incrementa el rendimiento, puesto que sólo se crea una instancia de un objeto una única vez, y luego se vuelve a utilizar para las invocaciones posteriores.

Conceptos relacionados:

- "Diseño de rutinas de automatización de OLE" en la página 194
- "Rutinas de automatización de OLE en BASIC y C++" en la página 198

Tareas relacionadas:

- "Creación de rutinas de automatización de OLE" en la página 195

Información relacionada:

- "Sentencia CREATE TYPE (Estructurado)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE FUNCTION (Escalar externa)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE PROCEDURE (Externo)" en la publicación *Consulta de SQL, Volumen 2*
- "Tipos de datos de SQL soportados en la automatización de OLE" en la página 197

Tipos de datos de SQL soportados en la automatización de OLE

DB2 maneja la conversión de tipos entre tipos de SQL y tipos de automatización de OLE. La tabla siguiente resume los tipos de datos soportados y la forma en que se correlacionan.

Tabla 28. Correlación de tipos de datos de SQL y de la automatización de OLE

Tipo de SQL	Tipo de la automatización de OLE	Descripción del tipo de la automatización de OLE
SMALLINT	short	Entero con signo de 16 bits
INTEGER	long	Entero con signo de 32 bits
REAL	float	Número de coma flotante IEEE de 32 bits
FLOAT o DOUBLE	double	Número de coma flotante IEEE de 64 bits
DATE	DATE	Número de días, fraccionario de coma flotante de 64 bits, desde el 30 de diciembre de 1899
TIME	DATE	
TIMESTAMP	DATE	
CHAR(*n*)	BSTR	Serie de longitud prefijada, descrita en la publicación *OLE Automation Programmer's Reference*.
VARCHAR(*n*)	BSTR	
LONG VARCHAR	BSTR	
CLOB(*n*)	BSTR	

Tabla 28. Correlación de tipos de datos de SQL y de la automatización de OLE (continuación)

Tipo de SQL	Tipo de la automatización de OLE	Descripción del tipo de la automatización de OLE
GRAPHIC(*n*)	BSTR	Serie de longitud prefijada, descrita en la publicación *OLE Automation Programmer's Reference*.
VARGRAPHIC(*n*)	BSTR	
LONG GRAPHIC	BSTR	
DBCLOB(*n*)	BSTR	
CHAR(*n*)	SAFEARRAY[car. sin signo]	Matriz Byte() de 1 dim de elementos de datos de 8 bits sin signo. (Los tipos SAFEARRAY se describen en la publicación *OLE Automation Programmer's Reference*.)
VARCHAR(*n*)	SAFEARRAY[car. sin signo]	
LONG VARCHAR	SAFEARRAY[car. sin signo]	
CHAR(*n*) FOR BIT DATA	SAFEARRAY[car. sin signo]	
VARCHAR(*n*) FOR BIT DATA	SAFEARRAY[car. sin signo]	
LONG VARCHAR FOR BIT DATA	SAFEARRAY[car. sin signo]	
BLOB(*n*)	SAFEARRAY[car. sin signo]	

Los datos que se pasan entre DB2 y las rutinas de automatización de OLE se pasan como llamadas por referencia. No están soportados los tipos de SQL tales como BIGINT, DECIMAL, DATALINK o LOCATORS ni los tipos de automatización de OLE tales como Boolean o CURRENCY que no aparecen listados en la tabla. Los datos gráficos y de tipo carácter correlacionados con BSTR se convierten de la página de códigos de la base de datos al esquema UCS-2. (UCS-2 se conoce también como Unicode, página de códigos 13488 de IBM). Al volver, los datos se vuelven a convertir de UCS-2 a la página de códigos de la base de datos. Estas conversiones se producen independientemente de la página de códigos de la base de datos. Si no están instaladas estas tablas de conversión de página de códigos, se recibirá el SQLCODE -332 (SQLSTATE 57017).

Conceptos relacionados:

- "Diseño de rutinas de automatización de OLE" en la página 194
- "Consideraciones sobre instancias de objetos y el área reutilizable y las rutinas de OLE" en la página 196
- "Rutinas de automatización de OLE en BASIC y C++" en la página 198

Tareas relacionadas:

- "Creación de rutinas de automatización de OLE" en la página 195

Rutinas de automatización de OLE en BASIC y C++

Puede implantar rutinas de automatización de OLE en cualquier lenguaje. En este apartado se muestra cómo implantar dichas rutinas, utilizando BASIC o C++ como dos lenguajes de ejemplo. La tabla siguiente muestra la correlación de los tipos de automatización de OLE con los tipos de datos en BASIC y C++.

Tabla 29. Correlación de tipos de datos de SQL y OLE con tipos de datos de BASIC y C++

Tipo de SQL	Tipo de la automatización de OLE	Tipo de BASIC	Tipo de C++
SMALLINT	short	Integer	short
INTEGER	long	Long	long
REAL	float	Single	float

Tabla 29. Correlación de tipos de datos de SQL y OLE con tipos de datos de BASIC y C++ (continuación)

Tipo de SQL	Tipo de la automatización de OLE	Tipo de BASIC	Tipo de C++
FLOAT o DOUBLE	double	Double	double
DATE, TIME, TIMESTAMP	DATE	Date	DATE
CHAR(*n*)	BSTR	String	BSTR
CHAR(*n*) FOR BIT DATA	SAFEARRAY[car. sin signo]	Byte()	SAFEARRAY
VARCHAR(*n*)	BSTR	String	BSTR
VARCHAR(*n*) FOR BIT DATA	SAFEARRAY[car. sin signo]	Byte()	SAFEARRAY
LONG VARCHAR	BSTR	String	BSTR
LONG VARCHAR FOR BIT DATA	SAFEARRAY[car. sin signo]	Byte()	SAFEARRAY
BLOB(*n*)	BSTR	String	BSTR
BLOB(*n*) FOR BIT DATA	SAFEARRAY[car. sin signo]	Byte()	SAFEARRAY
GRAPHIC(*n*), VARGRAPHIC(*n*), LONG GRAPHIC, DBCLOB(*n*)	BSTR	String	BSTR

Automatización de OLE en BASIC:

Para implantar rutinas de automatización de OLE en BASIC, tiene que utilizar los tipos de datos de BASIC que correspondan a los tipos de datos de SQL correlacionados con los tipos de automatización de OLE.

La declaración BASIC de la UDF de automatización de OLE, `bcounter`, tiene el aspecto siguiente:

```
Public Sub increment(output As Long, _
                     indicator As Integer, _
                     sqlstate As String, _
                     fname As String, _
                     fspecname As String, _
                     sqlmsg As String, _
                     scratchpad() As Byte, _
                     calltype As Long)
```

Automatización de OLE en C++:

La declaración C++ de la UDF de automatización de OLE, `increment`, es como sigue:

```
STDMETHODIMP Ccounter::increment (long   *output,
                                 short  *indicator,
                                 BSTR   *sqlstate,
                                 BSTR   *fname,
                                 BSTR   *fspecname,
                                 BSTR   *sqlmsg,
                                 SAFEARRAY **scratchpad,
                                 long   *calltype );
```

OLE soporta bibliotecas de tipos que describen las propiedades y los métodos de los objetos de automatización de OLE. Los objetos, las propiedades y los métodos expuestos se describen en el Lenguaje de descripción de objetos (Object Description Language - ODL). La descripción en ODL del método C++ anterior es la siguiente:

```
HRESULT increment ([out]    long  *output,
                   [out]    short *indicator,
                   [out]    BSTR  *sqlstate,
```

```
[in]     BSTR  *fname,
[in]     BSTR  *fspecname,
[out]    BSTR  *sqlmsg,
[in,out] SAFEARRAY (car. sin signo) *scratchpad,
[in]     long *calltype);
```

La descripción en ODL permite especificar si un parámetro es de entrada (in), de salida (out) o de entrada/salida (in,out). Para una rutina de automatización de OLE, los parámetros de entrada a la rutina y los indicadores de entrada se especifican como parámetros [in], y los parámetros de salida de la rutina y los indicadores de salida como parámetros [out]. Para los argumentos de cola de la rutina, `sqlstate` es un parámetro [out], `fname` y `fspecname` son parámetros [in], `scratchpad` es un parámetro [in,out] y `calltype` es un parámetro [in].

La automatización de OLE define el tipo de datos BSTR para manejar series. BSTR se define como puntero a OLECHAR: `typedef OLECHAR *BSTR`. Para asignar y liberar los BSTR, OLE impone la norma de que la rutina llamada libere un BSTR pasado como parámetro por referencia antes de asignar un nuevo valor al parámetro. Se aplica la misma norma para las matrices de bytes unidimensionales recibidas por la rutina llamada como `SAFEARRAY**`. Esta norma significa lo siguiente para DB2® y las rutinas de automatización de OLE:

- Parámetros [in]: DB2 asigna y libera los parámetros [in].
- Parámetros [out]: DB2 pasa un puntero con NULL. La rutina invocada debe asignar el parámetro [out], que DB2 liberará.
- Parámetros [in,out]: DB2 asigna inicialmente los parámetros [in,out]. La rutina invocada puede liberarlos y reasignarlos. Tal como para los parámetros [out], DB2 libera el parámetro final devuelto.

Todos los otros parámetros se pasan como punteros. DB2 asigna y gestiona la memoria referenciada.

La automatización de OLE proporciona un conjunto de funciones de manipulación de datos para tratar los BSTR y SAFEARRAY. Estas funciones se describen en la publicación *OLE Automation Programmer's Reference*.

La rutina en C++ siguiente devuelve los 5 primeros caracteres de un parámetro de entrada CLOB:

```
// UDF DDL: CREATE FUNCTION crunch (CLOB(5k)) RETURNS CHAR(5)

STDMETHODIMP Cobj::crunch (BSTR *in,          // CLOB(5K)
                           BSTR *out,         // CHAR(5)
                           short *indicator1,  // indicador de entrada
                           short *indicator2,  // indicador de salida
                           BSTR *sqlstate,    // puntero a NULL
                           BSTR *fname,       // puntero a un nombre de función
                           BSTR *fspecname,   // puntero a un nombre específico
                           BSTR *msgtext)     // puntero a NULL
  {
     // Asignar BSTR de 5 caracteres
     // y copiar 5 caracteres del parámetro de entrada

     // la salida es un parámetro [out] de tipo BSTR, es decir,
     // es un puntero a NULL y no es necesario liberar la memoria.
     // DB2 liberará el BSTR asignado.

     *out = SysAllocStringLen (*in, 5);
     return NOERROR;
  };
```

Un servidor de automatización de OLE se puede implantar como *creatable de un solo uso* o *creatable de varios usos*. Con creatable de un solo uso, cada cliente (es decir, un proceso FENCED de DB2) que conecte mediante `CoGetClassObject` con un objeto de automatización de OLE utilizará su propia instancia de una fábrica de clases, y, de ser necesario, ejecutará una nueva copia del servidor de automatización de OLE. Con creatable de varios usos, muchos clientes se conectan a la misma fábrica de clases. Es decir, a cada creación de una instancia de una fábrica de clases se le suministra una copia del servidor de OLE que ya está en funcionamiento, de haberlas. Si no existen copias del servidor de OLE en funcionamiento, automáticamente se inicia una copia para suministrar el objeto de clase. La elección entre una automatización de OLE de un solo uso o de varios usos corresponde al usuario, en el momento de implantar el servidor de automatización. Para obtener un rendimiento mejor, se recomienda un servidor de un solo uso.

Conceptos relacionados:

- "Automatización de Object Linking and Embedding (OLE) con Visual Basic" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Automatización de Object Linking and Embedding (OLE) con Visual C++" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Diseño de rutinas de automatización de OLE" en la página 194
- "Consideraciones sobre instancias de objetos y el área reutilizable y las rutinas de OLE" en la página 196

Tareas relacionadas:

- "Creación de rutinas de automatización de OLE" en la página 195

Información relacionada:

- "Sentencia CREATE FUNCTION" en la publicación *Consulta de SQL, Volumen 2*
- "Ejemplos de Object Linking and Embedding (OLE)" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Tipos de datos de SQL soportados en la automatización de OLE" en la página 197

Funciones de tabla de OLE DB definidas por el usuario

Los apartados siguientes describen cómo escribir funciones de tabla de OLE DB.

Funciones de tabla de OLE DB definidas por el usuario

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. El componente DBMS de 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 de OLE DB; un proveedor de OLE DB es un componente que expone interfaces de OLE DB. Existen dos clases de proveedores de OLE DB: los *proveedores de datos de OLE DB*, que poseen datos y los exponen en formato tabular como un conjunto de filas; y los *proveedores de servicio de OLE DB*, que no poseen sus propios datos, sino que encapsulan algunos servicios produciendo y consumiendo datos a través de las interfaces de OLE DB.

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.

DB2 es un consumidor de OLE DB que puede acceder a cualquier proveedor de servicio o de datos de OLE DB. El usuario puede realizar operaciones que incluyen GROUP BY, JOIN y UNION, sobre fuentes de datos que exponen sus datos a través de interfaces de OLE DB. Por ejemplo, puede definir una función de tabla de OLE DB para devolver una tabla de una base de datos Microsoft Access o de una agenda Microsoft Exchange y luego crear un informe que combine sin disrupciones datos procedentes de esta función de tabla de OLE DB con datos de su base de datos de DB2®.

La utilización de funciones de tabla OLE DB reduce el esfuerzo que hay que dedicar al desarrollo de aplicaciones al proporcionar un acceso integrado a cualquier proveedor de OLE DB. Para las funciones de tabla de automatización de C, Java™ y OLE, el desarrollador tiene que implementar la función de tabla, mientras que, en el caso de las funciones de tabla de OLE DB, un consumidor de OLE DB genérico integrado actúa como interfaz con cualquier proveedor de OLE DB para recuperar datos. Sólo es necesario registrar una función de tabla como LANGUAGE OLEDB y hacer referencia al proveedor de OLE DB y al conjunto de filas relevante como fuente de datos. No tiene que realizar ninguna programación de UDF para aprovechar las funciones de tabla de OLE DB.

Para utilizar las funciones de tabla de OLE DB con DB2 Universal Database, debe instalar OLE DB 2.0 o posteriores, que están disponibles en Microsoft, en la dirección `http://www.microsoft.com`. Si intenta invocar una función de tabla de OLE DB sin instalar primero OLE DB, DB2 emitirá el SQLCODE -465, SQLSTATE 58032, código de razón 35. Para conocer los requisitos del sistema y los proveedores de OLE DB disponibles para las fuentes de datos, consulte la documentación de la fuente de datos. Para ver la especificación de OLE DB, consulte la publicación *Microsoft OLE DB 2.0 Programmer's Reference and Data Access SDK*, Microsoft Press, 1998.

Restricciones para el uso de funciones de tabla de OLE DB: Las funciones de tabla de OLE DB no se pueden conectar a una base de datos de DB2.

Conceptos relacionados:

- "Funciones de tabla Object Linking and Embedding Database (OLE DB)" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Nombres de conjunto de filas completamente calificados" en la página 205

Tareas relacionadas:

- "Creación de una UDF para tablas OLE DB" en la página 203

Información relacionada:

- "Sentencia CREATE FUNCTION (Tabla externa OLE DB)" en la publicación *Consulta de SQL, Volumen 2*
- "Ejemplos de funciones de tabla de Object Linking and Embedding Database (OLE DB)" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Tipos de datos de SQL soportados en OLE DB" en la página 206

Creación de una UDF para tablas OLE DB

Para definir una función de tabla OLE DB con una sola sentencia CREATE FUNCTION, debe hacer lo siguiente:

- definir la tabla que devuelve el proveedor de OLE DB
- especificar LANGUAGE OLEDB
- identificar el conjunto de filas de OLE DB y proporcionar una serie de conexión del proveedor de OLE DB en la cláusula EXTERNAL NAME

Las fuentes de datos OLE DB exponen sus datos en forma tabular, llamada *conjunto de filas*. En un conjunto de filas, cada fila tiene un conjunto de columnas. La cláusula RETURNS TABLE sólo devuelve las columnas importantes para el usuario. El enlace de las columnas de la función de tabla con las columnas de un conjunto de filas de una fuente de datos OLE DB se basa en los nombres de columna. Si el proveedor de OLE DB es sensible a las mayúsculas y minúsculas, ponga los nombres de columna entre comillas; por ejemplo, `"UPPERcase"`.

La cláusula EXTERNAL NAME puede tomar cualquiera de las formas siguientes:

```
    'servidor!conjunto-filas'
o bien
    '!conjunto-filas!serie-conexión'
```

donde:

servidor
: identifica a un servidor registrado con la sentencia CREATE SERVER

conjunto-filas
: identifica a un conjunto de filas, o una tabla, expuesto por el proveedor de OLE DB; este valor debe estar vacío si la tabla tiene un parámetro de entrada que debe pasar al proveedor de OLE DB a través del texto del mandato.

serie-conexión
: contiene propiedades de inicialización necesarias para conectar con un proveedor de OLE DB. Para ver la sintaxis completa y la semántica de la serie de conexión, consulte el apartado "Data Link API of the OLE DB Core Components" en la publicación *Microsoft OLE DB 2.0 Programmer's Reference and Data Access SDK*, Microsoft Press, 1998.

Puede utilizar una *serie de conexión* en la cláusula EXTERNAL NAME de una sentencia CREATE FUNCTION o especificar la opción *CONNECTSTRING* en una sentencia CREATE SERVER.

Por ejemplo, puede definir una función de tabla OLE DB y devolver una tabla desde una base de datos Microsoft Access mediante las sentencias CREATE FUNCTION y SELECT siguientes:

```
CREATE FUNCTION orders ()
  RETURNS TABLE (orderid INTEGER, ...)
  LANGUAGE OLEDB
  EXTERNAL NAME '!orders!Provider=Microsoft.Jet.OLEDB.3.51;
                 Data Source=c:\msdasdk\bin\oledb\nwind.mdb';

SELECT orderid, DATE(orderdate) AS orderdate,
                DATE(shippeddate) AS shippeddate
FROM TABLE(orders()) AS t
WHERE orderid = 10248;
```

En lugar de colocar la serie de conexión en la cláusula EXTERNAL NAME, puede crear y utilizar un nombre de servidor. Por ejemplo, suponiendo que ha definido el servidor Nwind, puede utilizar la sentencia CREATE FUNCTION siguiente:

```
CREATE FUNCTION orders ()
  RETURNS TABLE (orderid INTEGER, ...)
  LANGUAGE OLEDB
  EXTERNAL NAME 'Nwind!orders';
```

Las funciones de tabla OLE DB también permiten especificar un parámetro de entrada de cualquier tipo de datos de serie de caracteres. Utilice el parámetro de entrada para pasar texto del mandato directamente al proveedor de OLE DB. Si define un parámetro de entrada, no proporcione el nombre de un conjunto de filas en la cláusula EXTERNAL NAME. DB2 pasa el texto del mandato al proveedor de OLE DB para su ejecución, y el proveedor de OLE DB devuelve a DB2 un conjunto de filas. Los nombres de columna y los tipos de datos del conjunto de filas resultante tienen que ser compatibles con la definición RETURNS TABLE de la sentencia CREATE FUNCTION. Puesto que el enlace de los nombres de columna del conjunto de filas se basa en la coincidencia de nombres de columna, se tiene que asegurar de denominar correctamente las columnas.

El ejemplo siguiente registra una función de tabla OLE DB, que recupera información de almacenamiento de una base de datos Microsoft SQL Server 7.0™. La serie de conexión se proporciona en la cláusula EXTERNAL NAME. Puesto que la función de tabla tiene un parámetro de entrada que debe pasar al proveedor de OLE DB a través del texto del mandato, el nombre del conjunto de filas no se especifica en la cláusula EXTERNAL NAME. El ejemplo de consulta pasa, en un mandato de SQL, texto que recupera información sobre los tres almacenes superiores de una base de datos SQL Server.

```
CREATE FUNCTION favorites (VARCHAR(600))
  RETURNS TABLE (store_id CHAR (4), name VARCHAR (41), sales INTEGER)
  SPECIFIC favorites
  LANGUAGE OLEDB
  EXTERNAL NAME '!!Provider=SQLOLEDB.1;Persist Security Info=False;
  User ID=sa;Initial Catalog=pubs;Data Source=WALTZ;
  Locale Identifier=1033;Use Procedure for Prepare=1;
  Auto Translate=False;Packet Size=4096;Workstation ID=WALTZ;
  OLE DB Services=CLIENTCURSOR;';

SELECT *
FROM TABLE (favorites (' select top 3 sales.stor_id as store_id,  ' ||
                       '        stores.stor_name as name,         ' ||
                       '        sum(sales. qty) as sales          ' ||
                       ' from sales, stores                       ' ||
                       ' where sales.stor_id = stores.stor_id     ' ||
                       ' group by sales.stor_id, stores.stor_name ' ||
                       ' order by sum(sales.qty) desc')) as f;
```

Conceptos relacionados:

- "Nombres de conjunto de filas completamente calificados" en la página 205
- "Funciones de tabla de OLE DB definidas por el usuario" en la página 201

Tareas relacionadas:

- "Instalación de DB2 Information Integrator y configuración del servidor para acceder a las fuentes de datos OLE DB" en la publicación *IBM DB2 Information Integrator Installation Guide*
- "Adición de fuentes de datos OLE DB a un servidor federado" en la publicación *IBM DB2 Information Integrator Guía de configuración de fuentes de datos*

Información relacionada:

- "Sentencia CREATE NICKNAME" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE SERVER" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE WRAPPER" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE FUNCTION (Tabla externa OLE DB)" en la publicación *Consulta de SQL, Volumen 2*
- "Ejemplos de funciones de tabla de Object Linking and Embedding Database (OLE DB)" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Tipos de datos de SQL soportados en OLE DB" en la página 206

Nombres de conjunto de filas completamente calificados

Es necesario identificar algunos conjuntos de filas en la cláusula EXTERNAL NAME mediante un *nombre completamente calificado*. Un nombre completamente calificado incorpora cualquiera de los elementos siguientes, o ambos:

- el nombre de catálogo asociado, que requiere la información siguiente:
  - si el proveedor soporta nombres de catálogo
  - dónde se debe situar el nombre de catálogo en el nombre completamente calificado
  - qué separador de nombres de catálogo se debe utilizar
- el nombre de esquema asociado, que requiere la información siguiente:
  - si el proveedor soporta nombres de esquema
  - qué separador de nombres de esquema se debe utilizar

Para obtener información sobre el soporte ofrecido por el proveedor de OLE DB respecto a los nombres de catálogo y de esquema, consulte la documentación de la información literal del proveedor de OLE DB.

Si `DBLITERAL_CATALOG_NAME` no es `NULL` en la información literal del proveedor, utilice un nombre de catálogo y el valor de `DBLITERAL_CATALOG_SEPARATOR` como separador. Para determinar si el nombre de catálogo va al principio o al final del nombre completamente calificado, consulte el valor de `DBPROP_CATALOGLOCATION` en el conjunto de propiedades `DBPROPSET_DATASOURCEINFO` del proveedor de OLE DB.

Si `DBLITERAL_SCHEMA_NAME` no es `NULL` en la información literal del proveedor, utilice un nombre de esquema y el valor de `DBLITERAL_SCHEMA_SEPARATOR` como separador.

Si los nombres contienen caracteres especiales o palabras clave coincidentes, encierre los nombres entre las comillas especificadas para el proveedor de OLE DB. Las comillas se definen en la información literal del proveedor de OLE DB como `DBLITERAL_QUOTE_PREFIX` y `DBLITERAL_QUOTE_SUFFIX`. Por ejemplo, en el EXTERNAL NAME siguiente, el conjunto de filas especificado incluye el nombre de catálogo *pubs* y el nombre de esquema *dbo* para una fila llamada *authors*, utilizándose la comilla " para encerrar los nombres.

```
EXTERNAL NAME '!"pubs"."dbo"."authors"!Provider=SQLOLEDB.1;...';
```

A fin de obtener más información sobre cómo construir nombres completamente calificados, consulte la publicación *Microsoft® OLE DB 2.0 Programmer's Reference and Data Access SDK*, Microsoft Press, 1998, así como la documentación correspondiente al proveedor de OLE DB.

Conceptos relacionados:

- "Funciones de tabla de OLE DB definidas por el usuario" en la página 201

Información relacionada:

- "Sentencia CREATE FUNCTION (Tabla externa OLE DB)" en la publicación *Consulta de SQL, Volumen 2*

Tipos de datos de SQL soportados en OLE DB

La tabla siguiente muestra cómo los tipos de datos de DB2 se correlacionan con los tipos de datos de OLE DB descritos en la publicación *Microsoft OLE DB 2.0 Programmer's Reference and Data Access SDK*, Microsoft Press, 1998. Utilice la tabla de correlación para definir las columnas RETURNS TABLE apropiadas en las funciones de tabla OLE DB. Por ejemplo, si se define una función de tabla OLE DB con una columna con tipo de datos INTEGER, DB2 solicita los datos al proveedor de OLE DB como DBTYPE_I4.

Para ver las correlaciones de los tipos de datos fuente del proveedor de OLE DB con los tipos de datos de OLE DB, consulte la documentación del proveedor de OLE DB. Para ver ejemplos de cómo los proveedores de ANSI SQL, Microsoft Access y Microsoft SQL Server pueden correlacionar sus tipos de datos respectivos con los tipos de datos de OLE DB, consulte la publicación *Microsoft OLE DB 2.0 Programmer's Reference and Data Access SDK*, Microsoft Press, 1998.

Tabla 30. Correlación de los tipos de datos de DB2 con OLE DB

Tipo de datos de DB2	Tipo de datos de OLE DB
SMALLINT	DBTYPE_I2
INTEGER	DBTYPE_I4
BIGINT	DBTYPE_I8
REAL	DBTYPE_R4
FLOAT/DOUBLE	DBTYPE_R8
DEC (p, s)	DBTYPE_NUMERIC (p, s)
DATE	DBTYPE_DBDATE
TIME	DBTYPE_DBTIME
TIMESTAMP	DBTYPE_DBTIMESTAMP
CHAR(N)	DBTYPE_STR
VARCHAR(N)	DBTYPE_STR
LONG VARCHAR	DBTYPE_STR
CLOB(N)	DBTYPE_STR
CHAR(N) FOR BIT DATA	DBTYPE_BYTES
VARCHAR(N) FOR BIT DATA	DBTYPE_BYTES
LONG VARCHAR FOR BIT DATA	DBTYPE_BYTES
BLOB(N)	DBTYPE_BYTES
GRAPHIC(N)	DBTYPE_WSTR
VARGRAPHIC(N)	DBTYPE_WSTR
LONG GRAPHIC	DBTYPE_WSTR
DBCLOB(N)	DBTYPE_WSTR

Nota: Las normas para la conversión de tipos de datos de OLE DB se definen en la publicación *Microsoft OLE DB 2.0 Programmer's Reference and Data Access SDK*, Microsoft Press, 1998. Por ejemplo:

- Para recuperar el tipo de datos de OLE DB DBTYPE_CY, los datos se pueden convertir al tipo de datos de OLE DB DBTYPE_NUMERIC(19,4), que está correlacionado con el tipo de datos de DB2 DEC(19,4).
- Para recuperar el tipo de datos de OLE DB DBTYPE_I1, los datos se pueden convertir al tipo de datos de OLE DB DBTYPE_I2, que está correlacionado con el tipo de datos de DB2 SMALLINT.
- Para recuperar el tipo de datos de OLE DB DBTYPE_GUID, los datos se pueden convertir al tipo de datos de OLE DB DBTYPE_BYTES, que se correlaciona con el tipo de datos de DB2 CHAR(12) FOR BIT DATA.

Conceptos relacionados:

• "Funciones de tabla de OLE DB definidas por el usuario" en la página 201

Tareas relacionadas:

• "Creación de una UDF para tablas OLE DB" en la página 203

Capítulo 5. Invocación de rutinas

Invocación de la rutina 209
Vías de acceso y nombres de rutina 211
Invocaciones de rutinas anidadas 213
Invocación de rutinas de 32 bits en un servidor de bases de datos de 64 bits 213
Consideraciones sobre la página de códigos de las rutinas 214
Invocación de procedimientos 215
Referencias a procedimientos 215
Selección de procedimiento 216
Llamada a procedimientos desde aplicaciones o rutinas externas 217
Llamada a procedimientos desde activadores o rutinas de SQL 219
Llamada a procedimientos desde el Procesador de línea de mandatos (CLP) 222
Invocación de funciones y métodos 224
Referencias a funciones 224
Selección de función 225
Tipos diferenciados como parámetros de UDF o método 227
Valores de LOB como parámetros de UDF . . . 228
Invocación de funciones escalares o métodos 229
Invocación de funciones de tabla definidas por el usuario 230

Invocación de la rutina

Una vez desarrollada y creada una rutina en la base de datos emitiendo la sentencia CREATE, si se han otorgado los privilegios adecuados sobre ésta al definidor de la rutina y al invocador de la rutina, se puede invocar la rutina.

Cada tipo de rutina cumple una finalidad distinta y se utiliza de forma distinta. Los requisitos previos para invocar rutinas son comunes, pero la implementación de la invocación difiere para cada una de ellas.

Requisitos previos de la invocación de la rutina:

- La rutina se debe haber creado en la base de datos utilizando la sentencia CREATE.
- En el caso de una rutina externa, el archivo de biblioteca o de clases debe estar instalado en la ubicación especificada por la cláusula EXTERNAL de la sentencia CREATE, pues, de lo contrario, se producirá un error (SQLCODE SQL0444, SQLSTATE 42724).
- El invocador de la rutina debe tener el privilegio EXECUTE sobre la rutina. Si el invocador no está autorizado a ejecutar la rutina, se producirá un error (SQLSTATE 42501).

Invocación de procedimientos:

Los procedimientos se invocan ejecutando la sentencia CALL con una referencia a un procedimiento.

La sentencia CALL permite la invocación de un procedimiento, el pase de parámetros al procedimiento y la recepción de parámetros devueltos por el procedimiento. Cualquier conjunto de resultados accesible devuelto por un procedimiento se puede procesar una vez que el procedimiento haya realizado su devolución de modo satisfactorio.

Los procedimientos se pueden invocar desde donde esté soportada la sentencia CALL, lo que incluye:

- Aplicaciones cliente
- Rutinas externas (procedimiento, UDF o método)

- Rutinas de SQL (procedimiento, UDF o método)
- Activadores (activadores BEFORE, AFTER o INSTEAD OF)
- Sentencias dinámicas compuestas
- Procesador de línea de mandatos (CLP)

Si elige invocar un procedimiento desde una aplicación cliente o desde una rutina externa, éstas pueden estar escritas en un lenguaje que no sea el del procedimiento. Por ejemplo, una aplicación cliente escrita en C++ puede utilizar la sentencia CALL para invocar un procedimiento escrito en Java™. Esto brinda a los programadores una gran flexibilidad para programar en el lenguaje que prefieran y para integrar partes de código escritas en lenguajes distintos.

Además, la aplicación cliente que invoca al procedimiento se puede ejecutar en un sistema operativo distinto de aquél en el que reside el procedimiento. Por ejemplo, una aplicación cliente que se ejecute en un sistema operativo Windows® puede utilizar la sentencia CALL para invocar un procedimiento que resida en un servidor de bases de datos Linux.

Según la procedencia de la invocación de un procedimiento, pueden existir diversas consideraciones adicionales.

Invocación de funciones:

Las funciones están pensadas para que se haga referencia a ellas dentro de sentencias de SQL.

Se puede hacer referencia a funciones incorporadas, a funciones de agregación con fuente y a funciones escalares definidas por el usuario en cualquier parte donde esté permitida una expresión dentro de una sentencia de SQL. Por ejemplo, dentro de la lista de selección de una consulta o dentro de la cláusula VALUES de una sentencia INSERT. Únicamente se puede hacer referencia a las funciones de tabla en la cláusula FROM. Por ejemplo, en la cláusula FROM de una consulta o de una sentencia de cambio de datos.

Invocación de métodos:

Los métodos son parecidos a las funciones escalares, excepto en que éstos se utilizan para proporcionar el comportamiento a los tipos estructurados. La invocación de métodos es como la invocación de funciones escalares definidas por el usuario, con la salvedad de que uno de los parámetros correspondientes al método debe ser el tipo estructurado sobre el que opera el método.

Tareas relacionadas con la invocación de una rutina:

Para invocar un tipo determinado de rutina:

- "Llamada a procedimientos desde aplicaciones o rutinas externas" en la página 217
- "Llamada a procedimientos desde activadores o rutinas de SQL" en la página 219
- Llamar a un procedimiento desde una aplicación CLI
- Llamar a un procedimiento desde el Procesador de línea de mandatos (CLP)
- "Invocación de funciones escalares o métodos" en la página 229
- "Invocación de funciones de tabla definidas por el usuario" en la página 230

Conceptos relacionados:

- "Rutinas en el desarrollo de aplicaciones" en la página 3
- "Consideraciones sobre la página de códigos de las rutinas" en la página 214
- "Vías de acceso y nombres de rutina" en la página 211
- "Invocaciones de rutinas anidadas" en la página 213

Tareas relacionadas:

- "Escritura de rutinas" en la página 36
- "Creación de rutinas en la base de datos" en la página 35
- "Invocación de rutinas de 32 bits en un servidor de bases de datos de 64 bits" en la página 213

Información relacionada:

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

Vías de acceso y nombres de rutina

El nombre calificado de un procedimiento almacenado o una UDF es `nombre-esquema.nombre-rutina`. Puede utilizar este nombre calificado en cualquier lugar en que haga referencia a un procedimiento almacenado o a una UDF. Por ejemplo:

```
SANDRA.BOAT_COMPARE    SMITH.FOO    SYSIBM.SUBSTR    SYSFUN.FLOOR
```

No obstante, también puede omitir el `nombre-esquema.`, en cuyo caso, DB2® intentará identificar el procedimiento almacenado o la UDF a la que se hace referencia. Por ejemplo:

```
BOAT_COMPARE    FOO    SUBSTR    FLOOR
```

El nombre calificado de un método es `nombre-esquema.tipo..nombre-método`.

El concepto de *vía de acceso SQL* es vital para la resolución por parte de DB2 de las referencias *no calificadas* que se producen cuando no se utiliza el `nombre-esquema`. La vía de acceso SQL es una lista ordenada de nombres de esquema. Proporciona un conjunto de esquemas para resolver las referencias no calificadas a procedimientos almacenados, UDF y tipos. En los casos en que una referencia coincide con un procedimiento almacenado, tipo o UDF en más de un esquema de la vía de acceso, se utiliza el orden de los esquemas de la vía de acceso para resolver esta coincidencia. La vía de acceso SQL se establece por medio de la función FUNCPATH en los mandatos de precompilación y enlace del SQL estático. En el SQL dinámico, se establece mediante la sentencia SET PATH. La vía de acceso SQL tiene el valor por omisión siguiente:

```
"SYSIBM","SYSFUN","SYSPROC", "ID"
```

Esto es aplicable al SQL tanto estático como dinámico, siendo que *ID* representa el ID de autorización de la sentencia actual.

Los nombres de rutina se pueden *sobrecargar*, lo que significa que múltiples rutinas, incluso del mismo esquema, pueden tener el mismo nombre. Múltiples funciones o métodos con el mismo nombre pueden tener el mismo número de parámetros, siempre y cuando los tipos de datos sean distintos. Esto no se aplica a los procedimientos almacenados, ya que varios procedimientos almacenados con el mismo nombre deben tener un número distinto de parámetros. Las instancias de distintos tipos de rutinas no se sobrecargan entre sí, a excepción de los métodos,

los cuales pueden sobrecargar las funciones. Para que un método sobrecargue una función, el método se debe registrar utilizando la cláusula WITH FUNCTION ACCESS.

Una función, un procedimiento almacenado y un método pueden tener *signaturas* idénticas y estar en el mismo esquema sin sobrecargarse entre sí. En el contexto de las rutinas, las signaturas son el nombre de rutina calificado que se concatena con los tipos de datos definidos de todos los parámetros en el orden en que se han definido.

Los métodos se invocan sobre instancias de su tipo estructurado asociado. Cuando se crea un subtipo, entre los atributos que éste hereda se encuentran los métodos definidos para el supertipo. Por ello, los métodos de un supertipo también se pueden ejecutar sobre las instancias de sus subtipos. Al definir un subtipo, se puede *alterar temporalmente* el método del supertipo. Alterar temporalmente un método significa volverlo a implementar específicamente para un subtipo determinado. Esto facilita el despacho dinámico de métodos (también conocido como polimorfismo), en que una aplicación ejecutará el método más específico según el tipo de instancia del tipo estructurado (por ejemplo, en qué lugar de la jerarquía de tipos estructurados está situado).

Cada tipo de rutina tiene su propio algoritmo de selección que tiene en cuenta los hechos de sobrecarga (y, en el caso de los métodos, la alteración temporal) y la vía de acceso SQL a fin de elegir la coincidencia más adecuada para cada referencia de rutina.

Conceptos relacionados:

- "Rutinas en el desarrollo de aplicaciones" en la página 3
- "Tipos estructurados definidos por el usuario" en la página 266
- "Despacho dinámico de métodos" en la página 272
- "Selección de función" en la página 225
- "Tipos de rutinas (procedimientos, funciones, métodos)" en la página 5
- "Selección de procedimiento" en la página 216

Tareas relacionadas:

- "Definición del comportamiento de los tipos estructurados" en la página 271

Información relacionada:

- "Funciones" en la publicación *Consulta de SQL, Volumen 1*
- "Sentencia CREATE FUNCTION" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE PROCEDURE" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE TYPE (Estructurado)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia SET PATH" en la publicación *Consulta de SQL, Volumen 2*
- "Mandato BIND" en la publicación *Consulta de mandatos*
- "Mandato PRECOMPILE" en la publicación *Consulta de mandatos*
- "Métodos" en la publicación *Consulta de SQL, Volumen 1*

Invocaciones de rutinas anidadas

En el contexto de las rutinas, el *anidamiento* hace referencia a una situación en que una rutina invoca a otra. Es decir, que el SQL emitido por una rutina puede hacer referencia a otra rutina, la cual puede emitir SQL que haga de nuevo referencia a otra rutina, y así sucesivamente. Si la serie de rutinas referida contiene una rutina referida anteriormente, se considera que es una situación de anidamiento *repetitivo*.

Puede utilizar el anidamiento y la repetición en las rutinas de DB2® con las restricciones siguientes:

16 niveles de anidamiento

Puede anidar invocaciones a rutinas hasta un máximo de 16 niveles de profundidad. Piense en un escenario en que la rutina A llama a la rutina B, y la rutina B llama a la rutina C. En este ejemplo, la ejecución de la rutina C está en el nivel de anidamiento 3. Son posibles trece niveles más de anidamiento.

Otras restricciones

Una rutina no puede llamar a una rutina de destino que esté catalogada con un nivel de acceso a datos SQL superior. Por ejemplo, una UDF creada con la cláusula CONTAINS SQL puede llamar a procedimientos almacenados creados con la cláusula CONTAINS SQL o la cláusula NO SQL. Sin embargo, esta rutina no podrá llamar a procedimientos almacenados creados con la cláusula READS SQL DATA o la cláusula MODIFIES SQL DATA (SQLCODE -577, SQLSTATE 38002). Es así porque el nivel de SQL del invocador no permite que tenga lugar ninguna operación de lectura o modificación (esto lo hereda la rutina invocada).

Otra limitación al anidar rutinas es que el acceso a tablas está restringido, a fin de evitar operaciones conflictivas de lectura y grabación entre las rutinas.

Conceptos relacionados:

- "Conflictos de datos cuando los procedimientos leen o graban en tablas" en la página 44
- "Consideraciones sobre seguridad para las rutinas" en la página 27

Invocación de rutinas de 32 bits en un servidor de bases de datos de 64 bits

Es posible invocar rutinas de 32 bits en un servidor de bases de datos de 64 bits. La primera vez que se invoca una rutina de 32 bits en un entorno como éste, tiene lugar una disminución del rendimiento. Las invocaciones posteriores del procedimiento almacenado de 32 bits funcionarán igual que una rutina de 64 bits equivalente.

Para los procedimientos Java, una JVM (Java Virtual Machine) de 32 bits puede funcionar en un servidor de bases de datos de 64 bits. Para las rutinas Java de 32 bits que utilicen esta JVM, no habrá una mayor actividad general de rendimiento. Una rutina de 64 bits comparable que utilice una JVM de 64 bits no se ejecutará más rápido. No obstante, una rutina Java de 32 bits que se ejecute en un servidor de bases de datos de 64 bits no se escalará bien porque la rutina se tiene que ejecutar en modalidad FENCED NOT THREADSAFE. Como consecuencia, cada invocación de dicha rutina requerirá su propia JVM.

Restricciones:

Las rutinas de 32 bits se deben registrar como FENCED y NOT THREADSAFE para funcionar en una instancia de 64 bits.

No es posible invocar rutinas de 32 bits en un servidor de bases de datos Linux/IA-64.

Procedimiento:

Para invocar rutinas existentes de 32 bits en un servidor de 64 bits:

1. Copie la clase o la biblioteca de la rutina en el directorio de rutinas de base de datos:
   - UNIX: sqllib/function
   - Windows: sqllib\function
2. Registre el procedimiento almacenado con la sentencia CREATE PROCEDURE.
3. Invoque al procedimiento almacenado con la sentencia CALL.

Conceptos relacionados:

- "Invocación de la rutina" en la página 209
- "Rutinas Java" en la página 181

Consideraciones sobre la página de códigos de las rutinas

Los datos de tipo carácter se pasan a las rutinas externas en la página de códigos implicada en la opción PARAMETER CCSID que se ha utilizado al crear la rutina. De forma similar, la base de datos supone que la serie de caracteres salida de la rutina utiliza la página de códigos implicada en la opción PARAMETER CCSID.

Cuando un programa cliente (que utiliza, por ejemplo, la página de códigos C) accede a una sección con una página de códigos diferente (por ejemplo, la página de códigos S) que invoca a una rutina que utiliza otra página de códigos (por ejemplo, la página de códigos R), se producen los sucesos siguientes:

1. Cuando se invoca una sentencia de SQL, los datos de tipo carácter de entrada se convierten, de la página de códigos de la aplicación cliente (C) a la página de códigos asociada con la sección (S). La conversión no tiene lugar para los BLOB ni los datos que se utilizarán como FOR BIT DATA.
2. Si la página de códigos de la rutina no es la de la sección, entonces, antes de que se invoque la rutina, los datos de tipo carácter de entrada (excepto BLOB y FOR BIT DATA) se convierten a la página de códigos de la rutina (R).

   Es muy recomendable que precompile, compile y enlace la rutina del servidor utilizando la página de códigos bajo la que se invocará la rutina (R). Esto puede no ser posible en todos los casos. Por ejemplo, se puede crear una base de datos Unicode en un entorno Windows®. No obstante, si el entorno Windows no tiene la página de códigos Unicode, se tendrá que precompilar, compilar y enlazar la aplicación que cree la rutina en una página de códigos de Windows. La rutina funcionará si la aplicación no tiene ningún carácter delimitador especial que el precompilador no comprenda.
3. Cuando la rutina finaliza, el gestor de bases de datos convierte todos los datos de tipo carácter de salida, de la página de códigos de la rutina (R) a la página de códigos de la sección (S), si es necesario. Si la rutina ha emitido un error durante su ejecución, el SQLSTATE y el mensaje de diagnóstico de la rutina

también se convertirán, de la página de códigos de la rutina a la página de códigos de la sección. No tiene lugar ninguna conversión para las series de caracteres BLOB y FOR BIT DATA.

4. Cuando la sentencia finaliza, los datos de tipo carácter de salida se convierten, de la página de códigos de la sección (S) a la página de códigos de la aplicación cliente (C) otra vez. La conversión no tiene lugar para los BLOB ni para los datos utilizados como FOR BIT DATA.

Utilizando la opción DBINFO en las sentencias CREATE FUNCTION, CREATE PROCEDURE y CREATE TYPE, se pasa a la rutina su página de códigos. Mediante el uso de esta información, una rutina sensible a la página de códigos se puede escribir de forma que funcione en muchas páginas de códigos distintas.

Conceptos relacionados:

- "Conversión de caracteres" en la publicación *Consulta de SQL, Volumen 1*
- "Obtención de valores de página de códigos" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente*
- "Página de códigos activa para precompilación y vinculación" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente*
- "Página de códigos activa para la ejecución de aplicaciones" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente*
- "Conversión de caracteres entre distintas páginas de códigos" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente*
- "Cuándo se produce la conversión de página de códigos" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente*
- "Sustitución de caracteres durante conversiones de páginas de códigos" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente*
- "Conversiones de páginas de códigos soportadas" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente*
- "Desarrollo de aplicaciones en situaciones de páginas de códigos distintas" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente*

Información relacionada:

- "Sentencia CREATE FUNCTION" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE PROCEDURE" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE TYPE (Estructurado)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE METHOD" en la publicación *Consulta de SQL, Volumen 2*
- "Supported territory codes and code pages" en la publicación *Administration Guide: Planning*
- "Conversion tables for code pages 923 and 924" en la publicación *Administration Guide: Planning*

Invocación de procedimientos

Referencias a procedimientos

Los procedimientos almacenados se invocan desde la sentencia CALL, en la que se les hace referencia con un nombre calificado (nombre de esquema y nombre del procedimiento almacenado), seguido de una lista de argumentos encerrados entre paréntesis. Un procedimiento almacenado también se puede invocar sin el nombre

de esquema, lo que da como resultado una elección entre posibles procedimientos almacenados de distintos esquemas con el mismo número de parámetros.

Cada parámetro pasado al procedimiento almacenado puede estar compuesto por una variable del lenguaje principal, un marcador de parámetro, una expresión o un NULL. A continuación, se indican las restricciones correspondientes a los parámetros de procedimiento almacenado:

- Los parámetros OUT e INOUT deben ser variables del lenguaje principal.
- No se pueden pasar NULL a los procedimientos almacenados Java™ a menos que el tipo de datos de SQL esté correlacionado con un tipo de clase de Java.
- No se pueden pasar NULL a los procedimientos almacenados PARAMETER STYLE GENERAL.

La posición de los argumentos es importante y se debe ajustar a la definición del procedimiento almacenado para que la semántica sea correcta. Tanto la posición de los argumentos como la definición del procedimiento almacenado se deben ajustar al propio cuerpo del procedimiento almacenado. DB2® no intenta barajar los argumentos para que coincidan mejor con una definición de procedimiento almacenado ni comprende la semántica de los parámetros de procedimiento almacenado individuales.

Conceptos relacionados:

- "Estilos de parámetros para rutinas externas" en la página 95

Tareas relacionadas:

- "Llamada a procedimientos desde el Procesador de línea de mandatos (CLP)" en la página 222
- "Calling stored procedures from CLI applications" en la publicación *CLI Guide and Reference, Volume 1*
- "Llamada a procedimientos desde activadores o rutinas de SQL" en la página 219
- "Llamada a procedimientos desde aplicaciones o rutinas externas" en la página 217

Información relacionada:

- "Sentencia CREATE PROCEDURE" en la publicación *Consulta de SQL, Volumen 2*
- "Sintaxis para pasar argumentos a rutinas escritas en C/C++, OLE o COBOL" en la página 97

Selección de procedimiento

Dada una invocación de procedimiento almacenado, el gestor de bases de datos debe decidir a cuál de los posibles procedimientos almacenados con el mismo nombre debe llamar. La resolución del procedimiento almacenado se realiza siguiendo los pasos explicados a continuación.

1. Se buscan todos los procedimientos almacenados del catálogo (SYSCAT.ROUTINES) que cumplan las condiciones siguientes:
   - Para las invocaciones en que se ha especificado el nombre de esquema (es decir, las referencias calificadas), el nombre de esquema y el nombre del procedimiento almacenado coinciden con el nombre de invocación.
   - Para las invocaciones en que el nombre de esquema no se ha especificado (es decir, las referencias no calificadas), el nombre del procedimiento almacenado

coincide con el nombre de invocación y tiene un nombre de esquema que coincide con uno de los esquemas de la vía de acceso SQL.
- El número de parámetros definidos coincide con la invocación.
- El invocador tiene el privilegio EXECUTE sobre el procedimiento almacenado.

2. Se elige el procedimiento almacenado cuyo esquema vaya primero en la vía de acceso SQL.

Si no quedan procedimientos almacenados candidatos después del primer paso, se devolverá un error (SQLSTATE 42884).

Conceptos relacionados:

- "Invocación de la rutina" en la página 209
- "Referencias a procedimientos" en la página 215

Tareas relacionadas:

- "Llamada a procedimientos desde activadores o rutinas de SQL" en la página 219
- "Llamada a procedimientos desde aplicaciones o rutinas externas" en la página 217

Llamada a procedimientos desde aplicaciones o rutinas externas

La invocación de un procedimiento (también denominado procedimiento almacenado) que encapsula lógica de una aplicación cliente o de una aplicación asociada con una rutina externa, se realiza fácilmente con un simple trabajo de configuración en la aplicación y mediante la sentencia CALL.

Requisitos previos:

El procedimiento se debe haber creado en la base de datos ejecutando la sentencia CREATE PROCEDURE.

Para los procedimientos externos, el archivo de biblioteca o de clases debe existir en la ubicación especificada por la cláusula EXTERNAL de la sentencia CREATE PROCEDURE.

El invocador del procedimiento debe tener los privilegios necesarios para ejecutar la sentencia CALL. El invocador del procedimiento, en este caso, es el ID de usuario que ejecuta la aplicación; no obstante, se aplican reglas especiales si se utiliza la opción de enlace DYNAMICRULES para la aplicación.

Procedimiento:

Se deben incluir ciertos elementos en la aplicación si desea que ésta invoque a un procedimiento. Al escribir la aplicación, debe realizar lo siguiente:

1. Declare, asigne e inicialice el almacenamiento para las estructuras de datos opcionales y las variables del lenguaje principal o marcadores de parámetros que necesita la sentencia CALL.

   Para ello:

   - Asigne una variable del lenguaje principal o marcador de parámetro que se utilice para cada parámetro del procedimiento.

- Inicialice las variables del lenguaje principal o los marcadores de parámetros correspondientes a los parámetros IN o INOUT.

2. Establezca una conexión de base de datos. Esto se consigue ejecutando una sentencia CONNECT TO en el lenguaje SQL intercalado o bien codificando una conexión de base de datos implícita.
3. Codifique la invocación del procedimiento. Después de la conexión de base de datos, puede codificar la invocación del procedimiento. Para ello, ejecute la sentencia CALL en el lenguaje SQL. Asegúrese de especificar una variable del lenguaje principal, constante o marcador de parámetro para cada parámetro IN, INOUT y OUT que espere el procedimiento.
4. Añada código para procesar los parámetros OUT e INOUT y los conjuntos de resultados. Este código debe ser posterior a la ejecución de la sentencia CALL.
5. Codifique una operación de base de datos COMMIT o ROLLBACK. Después de la sentencia CALL y de la evaluación de los valores de parámetro de salida o datos devueltos por el procedimiento, puede ser conveniente que la aplicación confirme o retrotraiga la transacción. Para ello, se incluye una sentencia COMMIT o ROLLBACK. Un procedimiento puede incluir una sentencia COMMIT o ROLLBACK, pero es una práctica recomendable que la gestión de transacciones se realice dentro de la aplicación cliente.

   Nota: Los procedimientos invocados desde una aplicación que haya establecido una conexión de tipo 2 con la base de datos no pueden emitir sentencias COMMIT ni ROLLBACK.
6. Desconéctese de la base de datos.
7. Prepare, compile, enlace y vincule la aplicación. Si la aplicación corresponde a una rutina externa, emita la sentencia CREATE a fin de crear la rutina y ubique la biblioteca de código externo en la vía de acceso de función adecuada para el sistema operativo de modo que el gestor de bases de datos la pueda encontrar.
8. Ejecute la aplicación o invoque la rutina externa. Se invocará la sentencia CALL que ha intercalado en la aplicación.

Nota: Puede codificar sentencias de SQL y lógica de rutina en cualquier momento entre los pasos 2 y 5.

Conceptos relacionados:

- "Invocación de la rutina" en la página 209
- "Selección de procedimiento" en la página 216
- "Referencias a procedimientos" en la página 215

Tareas relacionadas:

- "Llamada a procedimientos desde activadores o rutinas de SQL" en la página 219

Información relacionada:

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

Ejemplos relacionados:

- "spcall.c -- Call individual stored procedures"
- "spclient.c -- Call various stored procedures"
- "spclient.sqc -- Call various stored procedures (C)"
- "spclient.sqC -- Call various stored procedures (C++)"

- "SpClient.java -- Call a variety of types of stored procedures from SpServer.java (JDBC)"

Llamada a procedimientos desde activadores o rutinas de SQL

Llamar a un procedimiento desde una rutina de SQL, un activador o una sentencia dinámica compuesta es, básicamente, lo mismo. Se utilizan pasos iguales para implementar esta llamada. Este tema explica los pasos utilizando un escenario de activador. Se indica cualquier requisito previo o paso que difiera al llamar a un procedimiento desde una rutina o sentencia dinámica compuesta.

Requisitos previos:

- El procedimiento se debe haber creado en la base de datos ejecutando la sentencia CREATE PROCEDURE.
- Para los procedimientos externos, los archivos de biblioteca o clases se deben encontrar en la ubicación especificada por la cláusula EXTERNAL de la sentencia CREATE PROCEDURE.
- El creador de un activador que contenga una sentencia CALL debe tener el privilegio de ejecutar la sentencia CALL. Durante la ejecución, cuando se activa un activador, es en la autorización del creador del activador en la que se comprueba el privilegio para ejecutar la sentencia CALL. Un usuario que ejecute una sentencia dinámica compuesta que contenga una sentencia CALL debe tener el privilegio de ejecutar la sentencia CALL para ese procedimiento.
- Para invocar un activador, un usuario debe tener el privilegio de ejecutar la sentencia de cambio de datos asociada con el suceso activador. De forma similar, para invocar satisfactoriamente una rutina de SQL o sentencia dinámica compuesta, un usuario debe tener el privilegio EXECUTE sobre la rutina.

Restricciones:

Cuando se invoca un procedimiento desde un activador de SQL, una rutina de SQL o una sentencia dinámica compuesta, se aplican las restricciones siguientes:

- En los entornos de bases de datos particionadas, no es posible invocar procedimientos desde activadores o UDF de SQL.
- En las máquinas de multiprocesador simétrico (SMP), las llamadas de procedimiento desde activadores se ejecutan en un solo procesador.
- Un procedimiento que se ha de llamar desde un activador no debe contener una sentencia COMMIT ni una sentencia ROLLBACK que intente retrotraer la unidad de trabajo. La sentencia ROLLBACK TO SAVEPOINT está soportada dentro del procedimiento; no obstante, el punto de salvaguarda especificado debe estar en el procedimiento.
- Una retrotracción de una sentencia CALL desde un activador no retrotraerá las sentencias ejecutadas dentro del procedimiento.
- El procedimiento no debe modificar ninguna tabla federada. Esto significa que el procedimiento no debe contener una actualización (UPDATE) buscada de un apodo, una supresión (DELETE) buscada desde un apodo ni una inserción (INSERT) en un apodo.
- Los conjuntos de resultados especificados para el procedimiento no serán accesibles.

Los activadores BEFORE no se pueden crear si contienen una sentencia CALL que hace referencia a un procedimiento creado con un nivel de acceso de MODIFIES

SQL DATA. La ejecución de una sentencia CREATE TRIGGER para un activador de ese tipo fallará con el error (SQLSTATE 42987). Si desea más información sobre los niveles de acceso de SQL en las rutinas, consulte:

- "Niveles de acceso de SQL en rutinas de SQL" en la página 69
- "SQL en rutinas externas" en la página 110

Procedimiento:

Este apartado sobre el procedimiento explica cómo crear e invocar un activador que contenga una sentencia CALL. El SQL necesario para llamar a un procedimiento desde un activador es el mismo que se necesita para llamar a un procedimiento desde una rutina de SQL o sentencia dinámica compuesta.

1. Escriba una sentencia CREATE TRIGGER básica especificando los atributos deseados del activador. Consulte la forma de crear la sentencia CREATE TRIGGER.
2. En la parte de la acción activadora del activador, puede declarar variables de SQL para cualquier parámetro IN, INOUT u OUT especificado por el procedimiento. Consulte la forma de crear la sentencia DECLARE. Para conocer cómo se inicializan o establecen estas variables, consulte la sentencia de asignación. Las variables de transición de activador también se pueden utilizar como parámetros para un procedimiento.
3. En la parte de la acción activadora del activador, añada una sentencia CALL para el procedimiento. Especifique un valor o expresión para cada uno de los parámetros IN, INOUT y OUT del procedimiento.
4. Para los procedimientos de SQL, puede capturar opcionalmente el estado de retorno del procedimiento utilizando la sentencia GET DIAGNOSTICS. Con este fin, tendrá que emplear una variable de tipo entero que incluya el estado de retorno. Inmediatamente después de la sentencia CALL, añada simplemente una sentencia GET DIAGNOSTICS que asigne RETURN_STATUS a la variable de estado de retorno del activador local.
5. Una vez que haya terminado de escribir la sentencia CREATE TRIGGER, ahora puede ejecutarla de forma estática (desde una aplicación) o dinámica (desde el CLP o desde el Centro de control) para crear formalmente el activador en la base de datos.
6. Invoque el activador. Para ello, realice la ejecución sobre la sentencia de cambio de datos adecuada que corresponda al suceso activador.
7. Cuando la sentencia de cambio de datos se ejecuta sobre la tabla, se desencadenan los activadores adecuados definidos para dicha tabla. Cuando se ejecuta la acción activadora, se ejecutan las sentencias de SQL que contiene ésta, incluida la sentencia CALL.

Errores de ejecución:

Si el procedimiento intenta leer o grabar en una tabla en la que el activador también lee o graba, se puede emitir un error en el caso de que se detecte un conflicto de lectura o grabación. El conjunto de tablas que modifica el activador, incluida la tabla para la cual se ha definido el activador, se debe excluir de las tablas modificadas por el procedimiento.

Ejemplo: Llamada a un procedimiento de SQL desde un activador:

Este ejemplo ilustra cómo puede intercalar una sentencia CALL para invocar un procedimiento dentro de un activador y cómo se captura el estado de retorno de la

llamada de procedimiento utilizando la sentencia GET DIAGNOSTICS. El SQL siguiente crea las tablas necesarias, un procedimiento de lenguaje SQL PL y un activador AFTER.

```
CREATE TABLE T1 (c1 INT, c2 CHAR(2))@
CREATE TABLE T2 (c1 INT, c2 CHAR(2))@

CREATE PROCEDURE proc(IN val INT, IN name CHAR(2))
LANGUAGE SQL
DYNAMIC RESULTSETS 0
MODIFIES SQL DATA
BEGIN
  DECLARE rc INT DEFAULT 0;
  INSERT INTO TABLE T2 VALUES (val, name);
  GET DIANOSTICS rc = ROW_COUNT;
  IF ( rc > 0 ) THEN
      RETURN 0;
  ELSE
      RETURN -200;
  END IF;
END@

CREATE TRIGGER trig1 AFTER UPDATE ON t1
REFERENCING NEW AS n
FOR EACH ROW MODE DB2SQL
WHEN (n.c1 > 100);
BEGIN ATOMIC
   DECLARE rs INTEGER DEFAULT 0;
   CALL proc(n.c1, n.c2);
   GET DIANOSTICS rs = RETURN_STATUS;
   VALUES(CASE WHEN rc < 0 THEN RAISE_ERROR('70001', 'PROC CALL failed'));
END@
```

La emisión de la sentencia de SQL siguiente causará que el activador se desencadene y que se invoque el procedimiento.

```
UPDATE T1 SET c1 = c1+1 WHERE c2 = 'CA'@
```

Conceptos relacionados:

- “Niveles de acceso de SQL en rutinas de SQL” en la página 69
- “SQL en rutinas externas” en la página 110
- “Invocación de la rutina” en la página 209
- “Selección de procedimiento” en la página 216
- “Referencias a procedimientos” en la página 215

Tareas relacionadas:

- “Calling stored procedures from CLI applications” en la publicación *CLI Guide and Reference, Volume 1*
- “Llamada a procedimientos desde aplicaciones o rutinas externas” en la página 217

Información relacionada:

- “Sentencia CALL” en la publicación *Consulta de SQL, Volumen 2*
- “Sentencia CREATE PROCEDURE” en la publicación *Consulta de SQL, Volumen 2*
- “Sentencia CREATE TRIGGER” en la publicación *Consulta de SQL, Volumen 2*
- “Sentencia GET DIAGNOSTICS” en la publicación *Consulta de SQL, Volumen 2*

Llamada a procedimientos desde el Procesador de línea de mandatos (CLP)

Puede llamar a procedimientos almacenados utilizando la sentencia CALL desde la interfaz del procesador de línea de mandatos de DB2. El procedimiento almacenado al que se llame debe estar definido en las tablas de catálogo del sistema DB2.

Procedimiento:

Para llamar al procedimiento almacenado, primero conéctese a la base de datos:

```
db2 connect to sample user IDusuario using contraseña
```

donde *IDusuario* y *contraseña* son el ID de usuario y la contraseña de la instancia donde reside la base de datos sample.

Para utilizar la sentencia CALL, entre el nombre del procedimiento almacenado más cualquier valor de parámetro IN o INOUT, así como el signo '?' en calidad de contenedor de cada valor de parámetro OUT.

Los parámetros de un procedimiento almacenado se proporcionan en la sentencia CREATE PROCEDURE para dicho procedimiento en el archivo fuente del programa.

Ejemplo de procedimiento de SQL

Para obtener información sobre la creación de un procedimiento de SQL, consulte 'Creación de procedimientos de SQL' en el Capítulo 6. Procedimientos de SQL.

En el archivo whiles.db2, la sentencia CREATE PROCEDURE para la signatura de procedimiento DEPT_MEDIAN es la siguiente:

```
CREATE PROCEDURE DEPT_MEDIAN
(IN deptNumber SMALLINT, OUT medianSalary DOUBLE)
```

Para invocar este procedimiento, utilice la sentencia CALL en la que debe especificar el nombre del procedimiento y los argumentos de parámetro adecuados, que en este caso son el valor para el parámetro IN y un signo de interrogación, '?', para el valor del parámetro OUT. La sentencia SELECT del procedimiento utiliza el valor deptNumber de la columna DEPT de la tabla STAFF, por ello, a fin de obtener una salida significativa, el parámetro IN tiene que ser un valor correcto de la columna DEPT; por ejemplo, el valor "51":

```
db2 call dept_median (51, ?)
```

Nota: En las plataformas UNIX, los paréntesis tienen un significado especial para el shell de mandatos, por lo que deben ir precedidos del carácter "\" o se deben encerrar entre comillas, del modo siguiente:

```
db2 "call dept_median (51, ?)"
```

No sirven las comillas si se ha de utilizar la modalidad interactiva del procesador de línea de mandatos.

Después de ejecutar el mandato anterior, tendría que recibir un resultado como el siguiente:

```
Value of output parameters
--------------------------
Parameter Name  : MEDIANSALARY
Parameter Value : +1.76545000000000E+004

Return Status = 0
```

Ejemplo de procedimiento almacenado de C

Con el Procesador de línea de mandatos, también puede llamar a procedimientos almacenados creados a partir de los lenguajes principales soportados. En el directorio `samples/c` de UNIX y en el directorio `samples\c` de Windows, DB2 proporciona archivos de creación de procedimientos almacenados. La biblioteca compartida `spserver` contiene varios procedimientos almacenados que se pueden crear a partir del archivo fuente, `spserver.sqc`. El archivo `spcreate.db2` cataloga los procedimientos almacenados.

En el archivo `spcreate.db2`, la sentencia `CREATE PROCEDURE` para el procedimiento `MAIN_EXAMPLE` empieza así:

```
CREATE PROCEDURE MAIN_EXAMPLE (IN job CHAR(8),
                               OUT salary DOUBLE,
                               OUT errorcode INTEGER)
```

Para llamar a este procedimiento almacenado, tiene que colocar un valor CHAR para el parámetro IN, `job`, y un signo de interrogación, '?', para cada uno de los parámetros OUT. La sentencia SELECT del procedimiento utiliza el valor `job` de la columna JOB de la tabla EMPLOYEE, por ello, a fin de obtener una salida significativa, el parámetro IN tiene que ser un valor correcto de la columna JOB. El programa de ejemplo en C, `spclient`, que llama al procedimiento almacenado, utiliza `'DESIGNER'` para el valor de `JOB`. Se puede hacer lo mismo, del modo siguiente:

```
db2 "call MAIN_EXAMPLE ('DESIGNER', ?, ?)"
```

Después de ejecutar el mandato anterior, tendría que recibir un resultado como el siguiente:

```
Value of output parameters
--------------------------
Parameter Name  : SALARY
Parameter Value : +2.37312500000000E+004

Parameter Name  : ERRORCODE
Parameter Value : 0

Return Status = 0
```

Un `ERRORCODE` con el valor de cero indica un resultado satisfactorio.

Si se compara con el programa `spclient`, se puede observar que `spclient` ha formateado el resultado en decimal para facilitar la visualización:

```
CALL stored procedure named MAIN_EXAMPLE
Stored procedure returned successfully
Average salary for job DESIGNER =  23731.25
```

Tareas relacionadas:

- "Creación de procedimientos SQL" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Llamada a procedimientos SQL desde aplicaciones cliente" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

- "Invocación de procedimientos SQL mediante aplicaciones cliente en Windows" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Llamada a procedimientos desde activadores o rutinas de SQL" en la página 219
- "Llamada a procedimientos desde aplicaciones o rutinas externas" en la página 217

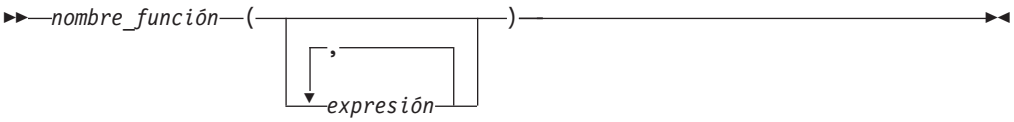
Ejemplos relacionados:

- "spclient.sqc -- Call various stored procedures (C)"
- "spcreate.db2 -- How to catalog the stored procedures contained in spserver.sqc (C)"
- "spserver.sqc -- Definition of various types of stored procedures (C)"
- "whiles.db2 -- To create the DEPT_MEDIAN SQL procedure "
- "whiles.sqc -- To call the DEPT_MEDIAN SQL procedure"

Invocación de funciones y métodos

Referencias a funciones

Cada referencia a una función, tanto si es una UDF como una función incorporada, contiene la sintaxis siguiente:



En el diagrama de sintaxis anterior, `nombre_función` puede ser un nombre de función calificado o no calificado. Puede haber de 0 a 90 argumentos, que son expresiones. Ejemplos de componentes que pueden constituir expresiones son:

- un nombre de columna, calificado o no calificado
- una constante
- una variable del lenguaje principal
- un registro especial
- un marcador de parámetros

La posición de los argumentos es importante y se debe ajustar a la definición de la función para que la semántica sea correcta. Tanto la posición de los argumentos como la definición de la función se deben ajustar al propio cuerpo de la función. DB2® no intenta barajar los argumentos para que coincidan mejor con una definición de función ni comprende la semántica de los parámetros de función individuales.

El uso de nombres de columna en las expresiones de argumento de UDF requiere que las referencias a las tablas que incluyen las columnas tengan el ámbito adecuado. Para las funciones de tabla referidas en una unión y que utilizan cualquier argumento que implique columnas de otra tabla o función de tabla, la tabla o función de tabla referida debe preceder a la función de tabla que contiene la referencia en la cláusula FROM.

Para poder utilizar marcadores de parámetros en funciones, no basta con codificar lo siguiente:

```
BLOOP(?)
```

Puesto que la lógica de selección de función no sabe en qué tipo de datos se puede convertir el argumento, no puede resolver la referencia. Se puede utilizar la especificación CAST para proporcionar el tipo del marcador de parámetros. Por ejemplo, INTEGER, y, luego, la lógica de selección de función podrá continuar:

```
BLOOP(CAST(? AS INTEGER))
```

He aquí algunos ejemplos válidos de invocaciones a funciones:

```
AVG(FLOAT_COLUMN)
BLOOP(COLUMN1)
BLOOP(FLOAT_COLUMN + CAST(? AS INTEGER))
BLOOP(:hostvar :indicvar)
BRIAN.PARSE(CHAR_COLUMN CONCAT USER, 1, 0, 0, 1)
CTR()
FLOOR(FLOAT_COLUMN)
PABLO.BLOOP(A+B)
PABLO.BLOOP(:hostvar)
"search_schema"(CURRENT FUNCTION PATH, 'GENE')
SUBSTR(COLUMN2,8,3)
SYSFUN.FLOOR(AVG(EMP.SALARY))
SYSFUN.AVG(SYSFUN.FLOOR(EMP.SALARY))
SYSIBM.SUBSTR(COLUMN2,11,LENGTH(COLUMN3))
SQRT(SELECT SUM(length*length)
     FROM triangles
     WHERE id= 'J522'
     AND legtype <> 'HYP')
```

Si algunas de las funciones anteriores son funciones de tabla, la sintaxis para hacerles referencia es ligeramente distinta de la presentada antes. Por ejemplo, si PABLO.BLOOP es una función de tabla, para hacerle referencia de forma correcta debe utilizar:

```
TABLE(PABLO.BLOOP(A+B)) AS Q
```

Tareas relacionadas:

- "Invocación de funciones escalares o métodos" en la página 229
- "Invocación de funciones de tabla definidas por el usuario" en la página 230

Información relacionada:

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

Selección de función

Para las referencias a funciones, tanto calificadas como no calificadas, el algoritmo de selección de función examina todas las *funciones pertinentes, tanto incorporadas como definidas por el usuario, que tienen:*

- El nombre indicado
- El mismo número de parámetros definidos que argumentos tiene la referencia a la función
- Cada uno de los parámetros idéntico o deducible a partir del tipo del argumento correspondiente.

Las funciones pertinentes son las funciones del esquema mencionado para una referencia calificada o las funciones de los esquemas de la vía de acceso SQL para

una referencia no calificada. El algoritmo busca una coincidencia exacta o, de no hallarla, la que más coincida entre estas funciones. La vía de acceso SQL se utiliza, únicamente en caso de una referencia no calificada, como factor decisorio si se encuentran dos coincidencias correctas idénticas en distintos esquemas.

Es posible anidar las referencias a funciones, incluso referencias a la misma función. Generalmente sucede así con las funciones incorporadas y con las UDF; sin embargo, existen algunas limitaciones cuando se ven implicadas funciones de columna.

Por ejemplo:

```
CREATE FUNCTION BLOOP (INTEGER) RETURNS INTEGER ...
CREATE FUNCTION BLOOP (DOUBLE) RETURNS INTEGER ...
```

Considere ahora la sentencia DML siguiente:

```
SELECT BLOOP( BLOOP(COLUMN1)) FROM T
```

Si column1 es una columna DECIMAL o DOUBLE, la referencia BLOOP interior se resuelve en la segunda BLOOP definida más arriba. Dado que esta BLOOP devuelve un INTEGER, la BLOOP exterior se resuelve en la primera BLOOP.

Alternativamente, si column1 es una columna SMALLINT o INTEGER, la referencia BLOOP interior se resuelve en la primera BLOOP definida más arriba. Dado que esta BLOOP devuelve un INTEGER, la BLOOP exterior también se resuelve en la primera BLOOP. En este caso, está contemplando referencias anidadas a la misma función.

Definiendo una función con el nombre de uno de los operadores de SQL, realmente se puede invocar una UDF utilizando *notación infix*. Por ejemplo, suponga que puede adjudicar algún significado al operador "+" para los valores que tienen el tipo diferenciado BOAT. Puede definir la UDF siguiente:

```
CREATE FUNCTION "+" (BOAT, BOAT) RETURNS ...
```

A continuación, puede escribir la sentencia válida de SQL siguiente:

```
SELECT BOAT_COL1 + BOAT_COL2
FROM BIG_BOATS
WHERE BOAT_OWNER = 'Nelson Mattos'
```

Pero también puede escribir la sentencia siguiente, igualmente válida:

```
SELECT "+"(BOAT_COL1, BOAT_COL2)
FROM BIG_BOATS
WHERE BOAT_OWNER = 'Nelson Mattos'
```

Observe que no tiene permitido sobrecargar los operadores condicionales incorporados, tales como >, =, LIKE, IN, etc., de esta manera.

Si desea una descripción más minuciosa de la selección de función, consulte el apartado sobre referencias a funciones en el tema de funciones listado entre los enlaces relacionados.

Conceptos relacionados:

• "Referencias a funciones" en la página 224

Tareas relacionadas:

• "Invocación de funciones escalares o métodos" en la página 229

- "Invocación de funciones de tabla definidas por el usuario" en la página 230

Información relacionada:
- "Funciones" en la publicación *Consulta de SQL, Volumen 1*
- "Sentencia CREATE FUNCTION" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE TYPE (Estructurado)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE METHOD" en la publicación *Consulta de SQL, Volumen 2*

Tipos diferenciados como parámetros de UDF o método

Las UDF y los métodos se pueden definir con tipos diferenciados como parámetros o como resultado. DB2 pasará el valor a la UDF o al método con el formato del tipo de datos fuente del tipo diferenciado.

Los valores de tipo diferenciado que se originan en una variable del lenguaje principal y que se utilizan como argumentos para una UDF que tiene su parámetro correspondiente definido como tipo diferenciado, **los debe difundir explícitamente el usuario al tipo diferenciado**. No existe ningún tipo de lenguaje principal para los tipos diferenciados. La tipificación estricta de DB2 lo requiere; de lo contrario, los resultados pueden ser ambiguos. Tome en consideración el tipo diferenciado BOAT que está definido sobre un BLOB y también la UDF BOAT_COST definida del modo siguiente:

```
CREATE FUNCTION BOAT_COST (BOAT)
  RETURNS INTEGER
  ...
```

En el fragmento siguiente de una aplicación en lenguaje C, la variable del lenguaje principal `:ship` contiene el valor BLOB que se debe pasar a la función BOAT_COST:

```
EXEC SQL BEGIN DECLARE SECTION;
   SQL TYPE IS BLOB(150K) ship;
EXEC SQL END DECLARE SECTION;
```

Las dos sentencias siguientes se resuelven correctamente en la función BOAT_COST, porque ambas difunden la variable del lenguaje principal `:ship` en el tipo BOAT:

```
... SELECT BOAT_COST (BOAT(:ship)) FROM ...
... SELECT BOAT_COST (CAST(:ship AS BOAT)) FROM ...
```

Si existen varios tipos diferenciados BOAT en la base de datos o varias UDF BOAT en otro esquema, debe tener cuidado con la vía de acceso SQL. De lo contrario, los resultados pueden ser ambiguos.

Conceptos relacionados:
- "Tipos definidos por el usuario (UDT) y objetos grandes (LOB)" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente*
- "Selección de función" en la página 225
- "Selección de procedimiento" en la página 216

Tareas relacionadas:
- "Pase de parámetros de tipo estructurado a rutinas externas" en la página 314
- "Valores de LOB como parámetros de UDF" en la página 228

Información relacionada:

- "Sentencia CREATE FUNCTION" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE PROCEDURE" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia SELECT" en la publicación *Consulta de SQL, Volumen 2*

Valores de LOB como parámetros de UDF

Las UDF se pueden definir con parámetros o resultados que tengan cualquiera de los tipos de LOB: BLOB, CLOB o DBCLOB. DB2 materializará el valor del LOB entero en el almacenamiento antes de invocar una función de este tipo, aunque la fuente del valor sea una variable del lenguaje principal de *localizador de LOB*. Por ejemplo, considere el fragmento siguiente de una aplicación en lenguaje C:

```
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS CLOB(150K) clob150K ;        /* var. leng. princ. LOB */
  SQL TYPE IS CLOB_LOCATOR clob_locator1;  /* var. leng. princ. localizador LOB */
   char                  string[40];       /* var. leng. princ. serie */
EXEC SQL END DECLARE SECTION;
```

Cualquiera de las variables del lenguaje principal `:clob150K` o `:clob_locator1` son válidas como argumento para una función cuyo parámetro correspondiente esté definido como `CLOB(500K)`. Por ejemplo, suponga que ha registrado una UDF del modo siguiente:

```
CREATE FUNCTION FINDSTRING (CLOB(500K, VARCHAR(200))
  ...
```

En el programa son válidas las dos invocaciones de FINDSTRING siguientes:

```
... SELECT FINDSTRING (:clob150K, :string) FROM ...
... SELECT FINDSTRING (:clob_locator1, :string) FROM ...
```

Los resultados o parámetros de UDF que tienen uno de los tipos de LOB se pueden crear con el modificador AS LOCATOR. En este caso, el valor de LOB entero no se materializa antes de la invocación. En cambio, se pasa un LOB LOCATOR a la UDF, la cual después puede utilizar SQL para manipular los bytes reales del valor de LOB.

También puede utilizar esta posibilidad en los resultados o parámetros de UDF que tienen un tipo diferenciado basado en un LOB. Observe que un argumento para una función de este tipo puede ser cualquier valor de LOB del tipo definido; no es necesario que sea una variable del lenguaje principal definida con uno de los tipos de LOCATOR. El uso de localizadores de variables del lenguaje principal como argumentos es completamente ortogonal al uso de AS LOCATOR en definiciones de resultados y parámetros de UDF.

Conceptos relacionados:

- "Tipos definidos por el usuario (UDT) y objetos grandes (LOB)" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente*
- "Selección de función" en la página 225
- "Selección de procedimiento" en la página 216

Tareas relacionadas:

- "Recuperación de un valor LOB con un localizador de LOB" en la página 238
- "Tipos diferenciados como parámetros de UDF o método" en la página 227

Información relacionada:

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

Invocación de funciones escalares o métodos

La invocación de funciones escalares incorporadas, funciones escalares definidas por el usuario y métodos es muy similar. Las funciones escalares y los métodos sólo se pueden invocar donde estén soportadas las expresiones dentro de una sentencia de SQL.

Requisitos previos:

- Para las funciones incorporadas, SYSIBM se debe encontrar en el registro especial CURRENT PATH. SYSIBM está en CURRENT PATH por omisión.
- Para las funciones escalares definidas por el usuario, la función se tiene que haber creado en la base de datos utilizando la sentencia CREATE FUNCTION o CREATE METHOD.
- Para las funciones escalares externas definidas por el usuario, el archivo de biblioteca o de clases asociado con la función se debe encontrar en la ubicación especificada por la cláusula EXTERNAL de la sentencia CREATE FUNCTION o CREATE METHOD.
- Para invocar una función definida por el usuario o método, un usuario debe tener el privilegio EXECUTE sobre la función o método. Si todos los usuarios van a utilizar la función o el método, el privilegio EXECUTE sobre la función o el método se puede otorgar a PUBLIC. Para obtener más información relativa a los privilegios, consulte la referencia a la sentencia CREATE específica.

Procedimiento:

Para invocar una UDF escalar o un método:

- Incluya una referencia a la función o método dentro de una expresión contenida en una sentencia de SQL, donde se van a procesar uno o más valores de entrada. Las funciones y métodos se pueden invocar en cualquier parte en la que una expresión sea válida. Los ejemplos de las partes en las que se puede hacer referencia a una UDF escalar o método incluyen la lista de selección de una consulta o una cláusula VALUES.

Por ejemplo, suponga que ha creado una función escalar definida por el usuario denominada TOTAL_SAL que añade el salario base y la bonificación juntos para cada fila de empleado en la tabla EMPLOYEE.

```
CREATE FUNCTION TOTAL_SAL
  (SALARY DECIMAL(9,2), BONUS DECIMAL(9,2))
  RETURNS DECIMAL(9,2)
  LANGUAGE SQL
  CONTAINS SQL
  NO EXTERNAL ACTION
  DETERMINISTIC
  RETURN SALARY+BONUS
```

A continuación mostramos una sentencia SELECT que hace uso de TOTAL_SAL:

```
SELECT LASTNAME, TOTAL_SAL(SALARY, BONUS) AS TOTAL
  FROM EMPLOYEE
```

Conceptos relacionados:

- "Referencias a funciones" en la página 224
- "Invocación de la rutina" en la página 209

- "Vías de acceso y nombres de rutina" en la página 211
- "Funciones escalares definidas por el usuario" en la página 15
- "Métodos" en la página 18

Tareas relacionadas:

- "Invocación de funciones de tabla definidas por el usuario" en la página 230

Información relacionada:

- "Sentencia SELECT" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE FUNCTION (Escalar de SQL, tabla o fila)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE FUNCTION (Escalar externa)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE FUNCTION (Con origen o plantilla)" en la publicación *Consulta de SQL, Volumen 2*

Ejemplos relacionados:

- "udfcli.sqc -- Call a variety of types of user-defined functions (C)"
- "udfemcli.sqc -- Call a variety of types of embedded SQL user-defined functions. (C)"
- "udfcli.sqC -- Call a variety of types of user-defined functions (C++)"
- "udfemcli.sqC -- Call a variety of types of embedded SQL user-defined functions. (C++)"
- "UDFcli.java -- Call the UDFs in UDFsrv.java (JDBC)"
- "UDFjcli.java -- Call the UDFs in UDFjsrv.java (JDBC)"
- "UDFcli.sqlj -- Call the UDFs in UDFsrv.java (SQLj)"
- "UDFjcli.sqlj -- Call the UDFs in UDFjsrv.java (SQLj)"

Invocación de funciones de tabla definidas por el usuario

Una vez que se escribe una función de tabla definida por el usuario y se registra con la base de datos, se puede invocar en la cláusula FROM de una sentencia SELECT.

Requisitos previos:

- La función de tabla se debe haber creado en la base de datos ejecutando CREATE FUNCTION.
- Para las funciones de tabla externas definidas por el usuario, el archivo de biblioteca o clases asociado con la función debe estar en la ubicación especificada por la cláusula EXTERNAL de CREATE FUNCTION.
- Para invocar una función de tabla definida por el usuario, un usuario debe tener el privilegio EXECUTE sobre la función. Si desea más información relacionada con los privilegios, consulte la referencia a CREATE FUNCTION.

Restricciones:

Para saber las restricciones de la invocación de funciones de tabla definidas por el usuario, consulte los temas sobre CREATE FUNCTION de los enlaces relacionados.

Procedimiento:

Para invocar una función de tabla definida por el usuario, haga referencia a la función en la cláusula FROM de una sentencia de SQL en la que se deba procesar un conjunto de valores de entrada. La referencia a la función de tabla debe ir precedida de la cláusula TABLE y debe aparecer entre corchetes.

Por ejemplo, la sentencia CREATE FUNCTION siguiente define una función de tabla que devuelve los empleados de un número de departamento especificado.

```
CREATE FUNCTION DEPTEMPLOYEES (DEPTNO VARCHAR(3))
RETURNS TABLE (EMPNO CHAR(6),
               LASTNAME VARCHAR(15),
               FIRSTNAME VARCHAR(12))
    LANGUAGE SQL
    READS SQL DATA
    NO EXTERNAL ACTION
    DETERMINISTIC
    RETURN
      SELECT EMPNO, LASTNAME, FIRSTNME FROM EMPLOYEE
        WHERE EMPLOYEE.WORKDEPT = DEPTEMPLOYEES.DEPTNO
```

A continuación, se muestra una sentencia SELECT que hace uso de DEPTEMPLOYEES:

```
SELECT EMPNO, LASTNAME, FIRSTNAME FROM TABLE(DEPTEMPLOYEES('A00')) AS D
```

Conceptos relacionados:

- "Referencias a funciones" en la página 224
- "Vías de acceso y nombres de rutina" en la página 211
- "Funciones escalares definidas por el usuario" en la página 17

Tareas relacionadas:

- "Valores de LOB como parámetros de UDF" en la página 228
- "Invocación de funciones escalares o métodos" en la página 229

Información relacionada:

- "Sentencia CREATE FUNCTION (Tabla externa OLE DB)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE FUNCTION (Escalar de SQL, tabla o fila)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE FUNCTION (Tabla externa)" en la publicación *Consulta de SQL, Volumen 2*

Ejemplos relacionados:

- "udfcli.sqc -- Call a variety of types of user-defined functions (C)"
- "udfemcli.sqc -- Call a variety of types of embedded SQL user-defined functions. (C)"
- "udfcli.sqC -- Call a variety of types of user-defined functions (C++)"
- "udfemcli.sqC -- Call a variety of types of embedded SQL user-defined functions. (C++)"
- "UDFcli.java -- Call the UDFs in UDFsrv.java (JDBC)"
- "UDFjcli.java -- Call the UDFs in UDFjsrv.java (JDBC)"
- "UDFcli.sqlj -- Call the UDFs in UDFsrv.java (SQLj)"
- "UDFjcli.sqlj -- Call the UDFs in UDFjsrv.java (SQLj)"

Parte 2. Objetos grandes, tipos diferenciados definidos por el usuario y activadores

Capítulo 6. Objetos grandes

Uso de objetos grandes 235
Localizadores de objetos grandes 236
Recuperación de un valor LOB con un localizador de LOB 238
Aplazamiento de la evaluación de las expresiones LOB 240
Variables de referencia a archivos de objetos grandes 242
Grabación de datos de una columna CLOB en un archivo de texto 244
Inserción de datos de un archivo de texto en una columna CLOB 245

Uso de objetos grandes

Los tipos de datos VARCHAR y VARGRAPHIC tienen un límite de 32 Kbytes de almacenamiento. Aunque esto puede ser suficiente para datos de texto de tamaño entre pequeño y mediano, a menudo las aplicaciones necesitan almacenar documentos grandes de texto. Es posible que también tengan que almacenar una amplia variedad de tipos de datos adicionales como audio, vídeo, dibujos, gráficos y texto combinados e imágenes. DB2® proporciona tres tipos de datos para almacenar estos objetos de datos como series de un máximo de dos (2) gigabytes (GB) de tamaño. Los tres tipos de datos de objetos grandes son: objetos grandes binarios (BLOB), objetos grandes de caracteres (CLOB) y objetos grandes de caracteres de doble byte (DBCLOB).

Nota: Los CLOB pueden contener caracteres tanto de un solo byte como de doble byte. Los DBCLOB pueden contener caracteres de cuatro bytes o de doble byte.

Cada tabla de DB2 puede tener una gran cantidad de datos LOB asociados. Aunque un solo valor de LOB no puede superar los 2 gigabytes, una sola fila puede contener hasta 24 gigabytes de datos LOB y una tabla puede contener hasta 4 terabytes de datos LOB.

Una ubicación de base de datos independiente almacena todos los valores de objetos grandes fuera de sus registros en la tabla. Hay un descriptor de objetos grandes para cada objeto grande de cada fila de una tabla. El descriptor de objetos grandes contiene información de control que se utiliza para acceder a los datos de objetos grandes almacenados en cualquier lugar del disco. El almacenamiento de datos de objetos grandes fuera de sus registros permite que los LOB puedan tener 2 GB de tamaño. El acceso al descriptor de objetos grandes ocasiona una pequeña cantidad de proceso general cuando se manipulan LOB. (Por motivos de almacenamiento y de rendimiento, no resulta recomendable colocar elementos de datos pequeños en LOB.)

El tamaño máximo de cada columna de objetos grandes es parte de la declaración del tipo de objeto grande en la sentencia CREATE TABLE. El tamaño máximo de una columna de objetos grandes determina el tamaño máximo de cualquier descriptor de LOB en dicha columna. Como resultado, también determina el modo en que las columnas de todos los tipos de datos se pueden ajustar en una sola fila. El espacio que utiliza el descriptor de LOB en la fila está comprendido entre 60 y 300 bytes, en función del tamaño máximo de la columna correspondiente.

La cláusula-opciones-lob de CREATE TABLE le permite registrar (o no) los cambios realizados en las columnas LOB. Esta cláusula también permite realizar una representación compacta para el descriptor de LOB (o no). Esto significa que puede

asignar sólo el espacio suficiente para almacenar el LOB o puede asignar espacio adicional para futuras operaciones de adición al LOB.

La cláusula-opciones-espaciotabla de CREATE TABLE le permite identificar un espacio de tabla LONG para almacenar los valores de columnas de tipos de datos de campo largo o LOB.

Con su tamaño potencialmente grande, los LOB pueden ralentizar el rendimiento del sistema de bases de datos significativamente cuando se incorporan a una base de datos o se extraen de la misma. Aunque DB2 no permite el registro de un valor LOB mayor que 1 GB, los valores LOB con tamaños cercanos a 1 GB pueden hacer que el registro de la base de datos alcance rápidamente su capacidad máxima. Se produce un error, SQLCODE -355 (SQLSTATE 42993), cuando se intenta registrar un LOB de más de 1 GB de tamaño. La cláusula-opciones-lob de las sentencias CREATE TABLE y ALTER TABLE permite a los usuarios desactivar el registro correspondiente a una determinada columna LOB. Aunque al definir la opción con el valor NOT LOGGED se mejora el rendimiento, los cambios realizados en los valores LOB tras la copia de seguridad más reciente se pierden durante una recuperación en avance.

Cuando seleccione un valor LOB, tendrá las siguientes opciones:

- Seleccionar el valor LOB entero en una variable del lenguaje principal. El valor LOB entero se copia del servidor en el cliente. Esto no resulta eficaz y a veces no es factible. Las variables del lenguaje principal utilizan el almacenamiento intermedio de memoria del cliente, el cual puede no tener una capacidad suficiente para albergar valores LOB mayores.
- Seleccione sólo un localizador de LOB en una variable del lenguaje principal. El valor LOB permanece en el servidor; el localizador de LOB se mueve al cliente. Si el valor LOB es muy grande y sólo se necesita como valor de entrada para una o más sentencias de SQL siguientes, es mejor conservar el valor en un localizador. El uso de un localizador elimina el tráfico de comunicación entre cliente y servidor necesario para transferir el valor LOB a la variable del lenguaje principal y de nuevo al servidor.
- Seleccionar el valor LOB entero en una variable de referencia de archivo. El valor LOB (o una parte de éste) se mueve a un archivo del cliente sin pasar por la memoria de la aplicación.

Conceptos relacionados:

- "Localizadores de objetos grandes" en la página 236
- "Variables de referencia a archivos de objetos grandes" en la página 242

Localizadores de objetos grandes

Un localizador de objetos grandes (o localizador de LOB) es una variable del lenguaje principal que tiene un valor de 4 bytes y representa un solo valor LOB del servidor de bases de datos. Los localizadores de LOB brindan a los usuarios un mecanismo mediante el cual pueden manipular fácilmente objetos muy grandes en programas de aplicación, sin necesidad de almacenar el valor LOB entero en la máquina cliente en la que se está ejecutando el programa de aplicación. Así, las sentencias posteriores pueden utilizar los localizadores para realizar operaciones sobre los datos sin tener que recuperar, necesariamente, el objeto grande entero. Las variables de localizador se utilizan para reducir las necesidades de almacenamiento para las aplicaciones que acceden a los LOB, y mejorar su rendimiento reduciendo el flujo de datos entre el cliente y el servidor.

Los localizadores de LOB son especialmente adecuados para varios escenarios de programación:

1. Cuando se mueve sólo una pequeña parte de un LOB mucho mayor a un programa cliente.
2. Cuando el LOB entero no puede caber en la memoria de la aplicación.
3. Cuando el programa necesita un valor LOB temporal procedente de una expresión LOB pero no necesita guardar el resultado.

Los localizadores de LOB también pueden representar el valor asociado a una expresión LOB. Por ejemplo, un localizador de LOB puede representar el valor asociado a:

```
SUBSTR( <lob 1> CONCAT <lob 2> CONCAT
        <lob 3>, <start>, <length> )
```

Es importante comprender que un localizador de LOB representa un valor, y no una fila o una ubicación de la base de datos. Una vez que se selecciona un valor en un localizador, no se puede realizar, sobre la fila o tabla original, ninguna operación que afecte al valor al que hace referencia el localizador. El valor asociado a un localizador es válido hasta que finaliza la transacción o hasta se libera de forma explícita el localizador, lo primero que se produce. Los localizadores no imponen copias adicionales de los datos para poder brindar esta función. En cambio, el mecanismo localizador almacena una descripción del valor LOB base. La materialización del valor (o de la expresión, como se ha mostrado anteriormente) LOB se aplaza hasta que se le asigna realmente alguna ubicación -- ya sea en el almacenamiento intermedio de un usuario, en forma de variable del lenguaje principal, o en el valor de un campo de otro registro de la base de datos.

Un localizador de LOB no es más que un mecanismo utilizado para hacer referencia a un valor LOB durante una transacción; no persiste más allá de la transacción en que se ha creado. La sentencia FREE LOCATOR libera un localizador del valor que tiene asociado. De forma similar, una operación de confirmación o de retrotracción libera todos los localizadores de LOB asociados con la transacción. Además, un localizador de LOB no es un tipo de base de datos; nunca se almacena en la base de datos y, como consecuencia, no puede participar en las vistas ni en las restricciones de comprobación. Sin embargo, y puesto que un localizador de LOB es la representación de un tipo de LOB en el cliente, existen diversos SQLTYPE para los localizadores de LOB, de forma que se puedan describir dentro de una estructura SQLDA utilizada por las sentencias FETCH, OPEN y EXECUTE. También se pueden pasar entre DB2® y las UDF.

Para variables de lenguaje principal normales en un programa de aplicación, cuando se selecciona un valor NULL en una variable de lenguaje principal, la variable del indicador se establece en -1, lo que significa que el valor es NULL. Sin embargo, en el caso de localizadores de LOB, el significado de las variables de indicador es ligeramente diferente. Puesto que la propia variable del lenguaje principal del localizador nunca puede tener el valor NULL, un valor negativo de variable de indicador indica que el valor LOB representado por el indicador LOB es NULL.

Conceptos relacionados:

- "Uso de objetos grandes" en la página 235

Tareas relacionadas:

- "Recuperación de un valor LOB con un localizador de LOB" en la página 238

- "Aplazamiento de la evaluación de las expresiones LOB" en la página 240

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++)"
- "dtLob.bas -- Get/set Large Objects (LOBs)"
- "DtLob.java -- How to use LOB data type (JDBC)"
- "DtLob.out -- HOW TO USE LOB DATA TYPE. Connect to 'sample' database using JDBC type 2 driver (JDBC)"
- "lobeval.sqb -- Demonstrates how to use a Large Object (LOB) (IBM COBOL)"
- "lobloc.sqb -- Demonstrates the use of LOB locators (IBM COBOL)"

Recuperación de un valor LOB con un localizador de LOB

Si necesita extraer datos de un LOB, puede utilizar localizadores de LOB. Ésta es una buena alternativa si el LOB al que se va a acceder es grande. Mediante el uso de localizadores, se evita el tener que transferir el LOB entero a un cliente cuando sólo se necesita un subconjunto de los datos del LOB.

En el ejemplo se utiliza SQL incorporado en C.

Procedimiento:

Para recuperar un valor LOB con un localizador de LOB:

1. Declare las variables del lenguaje principal del localizador de LOB:

```
EXEC SQL BEGIN DECLARE SECTION;
    char number[7];
    sqlint32 deptInfoBeginLoc;
    sqlint32 deptInfoEndLoc;
    SQL TYPE IS CLOB_LOCATOR resume;
    SQL TYPE IS CLOB_LOCATOR deptBuffer;
    short lobind;
    char buffer[1000]="";
    char userid[9];
    char passwd[19];
 EXEC SQL END DECLARE SECTION;
```

   En la sección de declaración de variables del lenguaje principal:
   - `number` contendrá el valor devuelto por `empno` en la sentencia SELECT que emitirá el cursor `c1`.
   - `deptInfoBeginLoc` y `deptInfoEnd` contendrán temporalmente los valores de localizador de LOB.
   - `resume` y `deptBuffer` son localizadores de LOB.
   - `lobind` se utiliza para indicar si la lectura del LOB es nula o no.
   - `buffer` contendrá los datos extraídos del LOB.
   - `userid` y `passwd` representan una combinación de id de usuario y contraseña, que son necesarios para que la aplicación conecte con una base de datos.
2. Conectar la aplicación con la base de datos.
3. Declare y abra un cursor:

```
EXEC SQL DECLARE c1 CURSOR FOR
   SELECT empno, resume FROM emp_resume WHERE resume_format='ascii'
   AND empno <> 'A00130';

EXEC SQL OPEN c1;
```

4. Lleve el valor LOB al localizador de variables del lenguaje principal.
   ```
   EXEC SQL FETCH c1 INTO :number, :resume :lobind;
   ```
5. Evalúe el localizador de LOB:
   a. Localice el inicio de la sección `Department Information`:
      ```
      EXEC SQL VALUES (POSSTR(:resume, 'Department Information'))
         INTO :deptInfoBeginLoc;
      ```
   b. Localice el inicio de la sección `Education` (final de `Department Information`):
      ```
      EXEC SQL VALUES (POSSTR(:resume, 'Education'))
         INTO :deptInfoEndLoc;
      ```
   c. Obtenga únicamente la sección `Department Information` utilizando SUBSTR:
      ```
      EXEC SQL VALUES(SUBSTR(:resume, :deptInfoBeginLoc,
        :deptInfoEndLoc - :deptInfoBeginLoc)) INTO :deptBuffer;
      ```
   d. Añada la sección `Department Information` a la variable `:buffer`:
      ```
      EXEC SQL VALUES(:buffer || :deptBuffer) INTO :buffer;
      ```
6. Libere los localizadores de LOB `resume` y `deptBuffer`:
   ```
   EXEC SQL FREE LOCATOR :resume, :deptBuffer;
   ```
7. Cierre el cursor:
   ```
   EXEC SQL CLOSE c1;
   ```
8. Finalice el programa.

Conceptos relacionados:

- "Uso de objetos grandes" en la página 235
- "Localizadores de objetos grandes" en la página 236

Tareas relacionadas:

- "Conexión de una aplicación con una base de datos" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente*
- "Finalización de un programa de aplicación" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente*
- "Aplazamiento de la evaluación de las expresiones LOB" en la página 240

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++)"
- "dtLob.bas -- Get/set Large Objects (LOBs)"
- "DtLob.java -- How to use LOB data type (JDBC)"
- "DtLob.out -- HOW TO USE LOB DATA TYPE. Connect to 'sample' database using JDBC type 2 driver (JDBC)"
- "lobeval.sqb -- Demonstrates how to use a Large Object (LOB) (IBM COBOL)"
- "lobloc.sqb -- Demonstrates the use of LOB locators (IBM COBOL)"

Aplazamiento de la evaluación de las expresiones LOB

No hay ningún movimiento de los bytes de un valor LOB hasta la asignación de una expresión LOB a un destino. Esto significa que un localizador de valor LOB utilizado con operadores y funciones de serie puede crear una expresión en la que la evaluación se posponga hasta el momento de la asignación. Esta técnica se conoce como aplazamiento de la evaluación de una expresión LOB.

El aplazamiento de evaluación permite a DB2 aumentar el rendimiento de E/S de LOB. Esto sucede porque el optimizador de la función de LOB intenta transformar las expresiones LOB en expresiones alternativas. Estas expresiones alternativas producen resultados equivalentes y habitualmente necesitan menos operaciones de E/S de disco.

En el ejemplo se utiliza SQL incorporado en C.

Procedimiento:

Para aplazar la evaluación de una expresión LOB:

1. Declare las variables del lenguaje principal del localizador de LOB:

```
EXEC SQL BEGIN DECLARE SECTION;
    sqlint32 hv_start_deptinfo;
    sqlint32 hv_start_educ;
    sqlint32 hv_return_code;
    SQL TYPE IS CLOB(5K) hv_new_section_buffer;
    SQL TYPE IS CLOB_LOCATOR hv_doc_locator1;
    SQL TYPE IS CLOB_LOCATOR hv_doc_locator2;
    SQL TYPE IS CLOB_LOCATOR hv_doc_locator3;
    char userid[9];
    char passwd[19];
EXEC SQL END DECLARE SECTION;
```

   En la sección de declaración de variables del lenguaje principal:
   - `hv_start_deptinfo`, `hv_return_code` y `hv_start_educ` contendrán temporalmente los valores de localizador de LOB.
   - `hv_new_section_buffer` contendrá los datos extraídos del LOB.
   - `hv_doc_locator1`, `hv_doc_locator2` y `hv_doc_locator3` son localizadores de LOB.
   - `userid` y `passwd` representan una combinación de id de usuario y contraseña, que son necesarios para que la aplicación conecte con una base de datos.
2. Conectar la aplicación con la base de datos.
3. Lleve el valor LOB al localizador de variables del lenguaje principal:

```
EXEC SQL SELECT resume INTO :hv_doc_locator1 FROM emp_resume
   WHERE empno = '000130' AND resume_format = 'ascii';
```

4. Manipule los datos LOB mediante localizadores. Estas cinco sentencias manipulan los datos LOB sin mover los datos reales contenidos en el campo de LOB.
   a. Utilice la función POSSTR para localizar el inicio de la sección `Department Information`:

```
EXEC SQL VALUES (POSSTR(:hv_doc_locator1, 'Department Information'))
   INTO :hv_start_deptinfo;
```

   b. Utilice la función POSSTR para localizar el inicio de la sección `Education`:

```
EXEC SQL VALUES (POSSTR(:hv_doc_locator1, 'Education'))
   INTO :hv_start_educ;
```

   c. Sustituya la sección `Department Information` por nada:

```
EXEC SQL VALUES (SUBSTR(:hv_doc_locator1, 1, :hv_start_deptinfo -1)
   || SUBSTR (:hv_doc_locator1, :hv_start_educ))
   INTO :hv_doc_locator2;
```

d. Mueva la sección `Department Information` a `hv_new_section_buffer` :

```
EXEC SQL VALUES (SUBSTR(:hv_doc_locator1, :hv_start_deptinfo,
   :hv_start_educ -:hv_start_deptinfo)) INTO :hv_new_section_buffer;
```

e. Añada la nueva sección al final. De hecho, no se hace más que mover la sección `Department Information` al final del resumen.

```
EXEC SQL VALUES (:hv_doc_locator2 || :hv_new_section_buffer)
   INTO :hv_doc_locator3;
```

5. Mueva los datos LOB al destino:

```
EXEC SQL INSERT INTO emp_resume
               VALUES ('A00130', 'ascii', :hv_doc_locator3);
```

La evaluación del LOB asignado al destino se pospone hasta esta sentencia. Es en este punto en el que, finalmente, se mueven los bytes del valor LOB.

6. Libere los localizadores de LOB `hv_doc_locator1`, `hv_doc_locator2` y `hv_doc_locator3`:

```
EXEC SQL FREE LOCATOR :hv_doc_locator1, :hv_doc_locator2,
                      : hv_doc_locator3;
```

7. Finalice el programa.

En este ejemplo, se ha solicitado un resumen particular (empno = '000130') de una tabla de resúmenes EMP_RESUME. La sección `Department Information` del resumen se ha copiado, cortado y luego añadido al final del resumen. Luego se ha insertado este nuevo resumen en la tabla EMP_RESUME. El resumen original de esta tabla no se ha modificado.

Los localizadores han permitido combinar y examinar el nuevo resumen sin mover ni copiar realmente ningún byte del resumen original. El movimiento de bytes no se produce hasta la asignación final; es decir, la sentencia INSERT -- y sólo se produce en el servidor.

Conceptos relacionados:

- "Uso de objetos grandes" en la página 235
- "Localizadores de objetos grandes" en la página 236

Tareas relacionadas:

- "Conexión de una aplicación con una base de datos" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente*
- "Finalización de un programa de aplicación" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente*
- "Recuperación de un valor LOB con un localizador de LOB" en la página 238

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++)"
- "dtLob.bas -- Get/set Large Objects (LOBs)"
- "DtLob.java -- How to use LOB data type (JDBC)"

- "DtLob.out -- HOW TO USE LOB DATA TYPE. Connect to 'sample' database using JDBC type 2 driver (JDBC)"
- "lobeval.sqb -- Demonstrates how to use a Large Object (LOB) (IBM COBOL)"

Variables de referencia a archivos de objetos grandes

Las variables de referencia a archivos LOB facilitan el movimiento de valores LOB desde el servidor de bases de datos a una aplicación cliente sin pasar por la memoria de la aplicación cliente. Las variables de referencia de archivo se parecen a las variables de lenguaje principal, excepto en que se utilizan para transferir datos a y desde archivos del cliente, y no para transferir datos a y desde los almacenamientos intermedios de memoria. Con este enfoque, las aplicaciones cliente no tienen que llamar a rutinas de programas de utilidad para leer y grabar archivos que utilizan variables de lenguaje principal (que tienen restricciones de tamaño) para llevar a cabo el movimiento de datos LOB.

Una variable de referencia de archivo representa (en lugar de contener) el archivo, del mismo modo que un localizador de LOB representa (en lugar de contener) el valor LOB. Las consultas, actualizaciones e inserciones de bases de datos pueden utilizar variables de referencia a archivos para almacenar o recuperar valores únicos de columnas LOB.

Las variables de referencia a archivos se utilizan para dirigir la entrada y salida de archivos para los LOB, y se pueden definir en todos los lenguajes de sistema principal. Puesto que no son tipos de datos nativos, se utilizan extensiones de SQL y los precompiladores generan las construcciones del lenguaje de sistema principal necesarias para representar cada variable.

Una variable de referencia a archivos tiene las propiedades siguientes:

1. Tipo de datos: BLOB, CLOB o DBCLOB. Esta propiedad se especifica cuando se declara la variable.
2. Nombre de archivo: El programa de aplicación lo debe especificar durante la ejecución. Es uno de los siguientes:
   - El nombre de la vía de acceso completa al archivo (aconsejable).
   - Un nombre de archivo relativo. Si se proporciona un nombre de archivo relativo, éste se añade a la vía de acceso actual del proceso del cliente. Dentro de una aplicación, sólo se debe hacer referencia a un archivo en una variable de referencia a archivos.
3. Longitud del nombre de archivo: El programa de aplicación la debe especificar durante la ejecución. Es la longitud del nombre de archivo (en bytes).
4. Opciones de archivo: Para que una aplicación haga uso de una variable de referencia a archivos, antes le debe asignar una de entre diversas opciones. Las opciones las establece un valor INTEGER de un campo de la estructura de la variable de referencia a archivos. Se debe especificar una de las opciones de archivo siguientes para cada variable de referencia a archivos:

Opción de archivo (por lenguaje)	Dirección	Descripción
C: SQL_FILE_READ COBOL: SQL-FILE-READ FORTRAN: sql_file_read	entrada	Se trata de un archivo normal que se puede abrir, leer y cerrar.

Opción de archivo (por lenguaje)	Dirección	Descripción
C: SQL_FILE_CREATE COBOL: SQL-FILE-CREATE FORTRAN: sql_file_create	salida	Crear un nuevo archivo. Si el archivo ya existe, se devuelve un error.
C: SQL_FILE_OVERWRITE COBOL: SQL-FILE-OVERWRITE FORTRAN: sql_file_overwrite	salida	Si existe un archivo con el nombre especificado, se sobregraba; de lo contrario, se crea un archivo nuevo.
C: SQL_FILE_APPEND COBOL: SQL-FILE-APPEND FORTRAN: sql_file_append	salida	Si existe un archivo con el nombre especificado, se le añade la salida; de lo contrario, se crea un archivo nuevo.

5. Longitud de datos: No se utiliza en la entrada. En la salida, la implementación establece la longitud de los datos (en bytes) como la longitud de los nuevos datos grabados en el archivo.

Para variables de lenguaje principal normales en un programa de aplicación, cuando se selecciona un valor NULL en una variable de lenguaje principal, la variable del indicador se establece en -1, lo que significa que el valor es NULL. Sin embargo, en el caso de variables de referencia a archivos, el significado de las variables de indicador es ligeramente diferente. Puesto que la propia variable de referencia a archivos nunca puede ser NULL, un valor negativo de la variable de indicador indica que el valor LOB representado por la variable de referencia a archivos es NULL.

Desde el sistema en el que se ejecuta el programa se tiene que poder acceder al archivo al que hace referencia la variable de referencia de archivo, aunque no hace falta que este resida en dicho sistema. Para un procedimiento almacenado, este sería el servidor.

En un entorno Extended UNIX® Code (EUC), se da por supuesto que los archivos a los que apuntan las variables de referencia a archivos DBCLOB contienen caracteres EUC válidos apropiados para almacenarlos en una columna gráfica, y que nunca contienen caracteres UCS-2.

Si se utiliza una variable de referencia de archivo LOB en una sentencia OPEN, el archivo asociado con la variable de referencia de archivo LOB no se debe suprimir hasta que se cierre el cursor.

Conceptos relacionados:

- "Uso de objetos grandes" en la página 235

Tareas relacionadas:

- "Grabación de datos de una columna CLOB en un archivo de texto" en la página 244
- "Inserción de datos de un archivo de texto en una columna CLOB" en la página 245

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++)"
- "dtLob.bas -- Get/set Large Objects (LOBs)"
- "DtLob.java -- How to use LOB data type (JDBC)"
- "DtLob.out -- HOW TO USE LOB DATA TYPE. Connect to 'sample' database using JDBC type 2 driver (JDBC)"
- "lobfile.sqb -- Demonstrates the use of LOB file handles (IBM COBOL)"

Grabación de datos de una columna CLOB en un archivo de texto

Si tiene necesidad de acceder a datos de una columna CLOB que se encuentra fuera de la base de datos, grábela en un archivo de texto.

El ejemplo contenido en el procedimiento utiliza SQL incorporado en C. En este ejemplo, se selecciona (SELECT) un resumen particular (empno = '000130') de una columna CLOB y se coloca en un archivo de texto.

Procedimiento:

Para grabar datos de una columna CLOB en un archivo de texto:

1. Declare la variable del lenguaje principal CLOB FILE REFERENCE:

   ```
   EXEC SQL BEGIN DECLARE SECTION;
      SQL TYPE IS CLOB_FILE resume;
      char userid[9];
      char passwd[19];
      short lobind;
   EXEC SQL END DECLARE SECTION;
   ```

   En la sección de declaración de variables del lenguaje principal:
   - `resume` representa el archivo que va a contener los datos extraídos de la columna CLOB.
   - `userid` y `passwd` representan una combinación de id de usuario y contraseña, que son necesarios para que la aplicación conecte con una base de datos.
2. Conectar la aplicación con la base de datos.
3. Configure la variable del lenguaje principal CLOB FILE REFERENCE:

   ```
   strcpy (resume.name, "RESUME.TXT");
   resume.name_length = strlen("RESUME.TXT");
   resume.file_options = SQL_FILE_OVERWRITE;
   ```

   En la descripción de la vía de acceso proporcionada en la función strcpy:
   - `RESUME.TXT` es el nombre del archivo cuyos datos se van a insertar en la tabla.
4. Seleccione (SELECT) los datos del campo de resumen de la columna CLOB en el archivo de texto especificado.

   ```
   EXEC SQL SELECT resume INTO :resume :lobind FROM emp_resume
      WHERE resume_format='ascii' AND empno='000130';
   ```
5. Finalice el programa.

Conceptos relacionados:

- "Localizadores de objetos grandes" en la página 236
- "Variables de referencia a archivos de objetos grandes" en la página 242

Tareas relacionadas:

- "Conexión de una aplicación con una base de datos" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente*
- "Finalización de un programa de aplicación" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente*
- "Inserción de datos de un archivo de texto en una columna CLOB" en la página 245

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++)"
- "dtLob.bas -- Get/set Large Objects (LOBs)"
- "DtLob.java -- How to use LOB data type (JDBC)"
- "DtLob.out -- HOW TO USE LOB DATA TYPE. Connect to 'sample' database using JDBC type 2 driver (JDBC)"
- "lobfile.sqb -- Demonstrates the use of LOB file handles (IBM COBOL)"

Inserción de datos de un archivo de texto en una columna CLOB

Si tiene necesidad de que la base de datos procese datos CLOB que existen actualmente en un archivo de texto, insértelo en una columna CLOB.

En el ejemplo se utiliza SQL incorporado en C en un sistema de archivos basado en UNIX.

Procedimiento:

Para insertar datos de un archivo de texto en una columna CLOB:

1. Declare la variable del lenguaje principal CLOB FILE REFERENCE:

   ```
   EXEC SQL BEGIN DECLARE SECTION;
     SQL TYPE IS CLOB_FILE hv_text_file;
   EXEC SQL END DECLARE SECTION;
   ```

   `hv_text_file` representa un archivo.
2. Conectar la aplicación con la base de datos.
3. Configure la variable del lenguaje principal CLOB FILE REFERENCE:

   ```
   strcpy(hv_text_file.name, "/u/userid/dirname/filnam.1");
   hv_text_file.name_length = strlen("/u/userid/dirname/filnam.1");
   hv_text_file.file_options = SQL_FILE_READ;
   ```

   En la descripción de la vía de acceso proporcionada en la función strcpy:
   - `userid` representa el directorio correspondiente a uno de sus usuarios.
   - `dirname` representa un subdirectorio que pertenece a "idusuario".
   - `filnam.1` es el nombre del archivo cuyos datos se van a insertar en la tabla.
   - `clobtab` es el nombre de la tabla con el tipo de datos CLOB.
4. Inserte datos de `hv_text_file` en la tabla CLOB.

   ```
   EXEC SQL INSERT INTO CLOBTAB
     VALUES(:hv_text_file);
   ```
5. Finalizar el programa.

Conceptos relacionados:
- "Localizadores de objetos grandes" en la página 236

- "Variables de referencia a archivos de objetos grandes" en la página 242

Tareas relacionadas:

- "Conexión de una aplicación con una base de datos" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente*
- "Finalización de un programa de aplicación" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente*
- "Grabación de datos de una columna CLOB en un archivo de texto" en la página 244

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++)"
- "dtLob.bas -- Get/set Large Objects (LOBs)"
- "DtLob.java -- How to use LOB data type (JDBC)"
- "DtLob.out -- HOW TO USE LOB DATA TYPE. Connect to 'sample' database using JDBC type 2 driver (JDBC)"
- "lobfile.sqb -- Demonstrates the use of LOB file handles (IBM COBOL)"

Capítulo 7. Tipos diferenciados definidos por el usuario

Tipos definidos por el usuario 247
Tipos diferenciados definidos por el usuario . . . 247
Tipificación estricta en tipos diferenciados definidos por el usuario 249
Creación de tipos diferenciados 250
Creación de tablas con columnas basadas en tipos diferenciados 251
Eliminación de tipos definidos por el usuario . . 252
Creación de tipos diferenciados basados en la moneda 253
Creación de un tipo diferenciado para formularios de solicitud de trabajo cumplimentados 254
Creación de tablas para hacer un seguimiento de las ventas internacionales 255
Creación de una tabla para almacenar formularios de solicitud de trabajo cumplimentados 255
Manipulación de tipos diferenciados. 256
Manipulación de tipos diferenciados. 256
Conversión del tipo de datos entre tipos diferenciados 257
Realización de comparaciones que implican a tipos diferenciados 258
Realización de comparaciones entre tipos diferenciados y constantes 259
Realización de asignaciones que implican a tipos diferenciados en SQL incorporado . . . 260
Realización de asignaciones que implican a tipos diferenciados en SQL dinámico 260
Realización de asignaciones que implican a distintos tipos diferenciados 261
Realización de operaciones UNION sobre columnas con tipo diferenciado 262
Definición de UDF con fuente para tipos diferenciados 262

Tipos definidos por el usuario

Un tipo definido por el usuario (UDT) es un tipo de datos que se deduce de los tipos de datos existentes pero, no obstante, se considera separado e incompatible respecto de ellos. Los UDT le permiten ampliar los tipos incorporados de que ya se dispone en DB2® y crear sus propios tipos de datos personalizados.

Existen dos clasificaciones de tipos definidos por el usuario:

- tipo diferenciado: comparte una representación común con tipos de datos incorporados.
- tipo estructurado: permite la representación de una secuencia de atributos con nombre, cada uno de los cuales tiene un tipo. Un tipo estructurado puede ser un subtipo de otro tipo estructurado (denominado supertipo), definiendo una jerarquía de tipos.

Conceptos relacionados:

- "Tipos diferenciados definidos por el usuario" en la página 247
- "Tipos estructurados definidos por el usuario" en la página 266

Tareas relacionadas:

- "Creación de tipos diferenciados" en la página 250

Tipos diferenciados definidos por el usuario

Los tipos diferenciados son tipos definidos por el usuario que se basan en tipos de datos incorporados en DB2®. Internamente, un tipo diferenciado comparte su representación con un tipo existente (el tipo fuente), pero se considera un tipo separado e incompatible.

Por ejemplo, los tipos diferenciados pueden representar diversas monedas, como por ejemplo el dólar americano y el dólar canadiense. Ambos tipos se representan internamente (y en el programa del lenguaje principal) como el tipo incorporado

en el que se han definido estas monedas. Por ejemplo, si se definen ambas monedas como DECIMAL, se representan como tipos de datos decimales en el sistema.

DB2 también dispone de tipos incorporados para almacenar y manipular objetos grandes. El tipo diferenciado se puede basar en uno de estos tipos de datos para objetos grandes (LOB), que se pueden querer utilizar para cosas tales como una corriente de audio o de vídeo. El ejemplo siguiente ilustra la creación de un tipo diferenciado denominado AUDIO:

```
CREATE DISTINCT TYPE AUDIO AS BLOB (1M)
```

Aunque AUDIO tiene la misma representación que el tipo de datos incorporado BLOB, se considera que es un tipo separado que no es comparable con un BLOB ni con ningún otro tipo. Esto permite la creación de funciones escritas específicamente para AUDIO y asegura que dichas funciones no se aplicarán a ningún otro tipo.

Los tipos diferenciados tienen asociadas varias ventajas:

1. Capacidad de ampliación: Mediante la definición de nuevos tipos, puede incrementar el conjunto de tipos proporcionados por DB2 para soportar las aplicaciones.
2. Flexibilidad: Puede especificar cualquier semántica y comportamiento para el tipo nuevo utilizando funciones definidas por el usuario (UDF) con el fin de aumentar la diversidad de tipos disponibles en el sistema.
3. Comportamiento coherente: La tipificación estricta asegura que los tipos diferenciados se comportarán adecuadamente. Garantiza que sólo las funciones definidas en su tipo diferenciado se pueden aplicar a instancias del tipo diferenciado.
4. Encapsulación: El conjunto de funciones y operadores que puede aplicar a los tipos diferenciados define el comportamiento de los tipos diferenciados. Esto proporciona flexibilidad en la implantación puesto que la ejecución de aplicaciones no depende de la representación interna que elija para el tipo.
5. Rendimiento: Los tipos diferenciados se integran muy bien en el gestor de bases de datos. Puesto que los tipos diferenciados se representan internamente del mismo modo que los tipos de datos incorporados, comparten el mismo código eficaz que se utiliza para implantar componentes tales como funciones incorporadas, operadores de comparación e índices para los tipos de datos incorporados.

Los tipos diferenciados se identifican mediante identificadores calificados. Si no se utiliza el nombre de esquema para calificar el nombre del tipo diferenciado al utilizarlo en sentencias distintas de CREATE DISTINCT TYPE, DROP DISTINCT TYPE o COMMENT ON DISTINCT TYPE, se busca en la vía de acceso de SQL, en secuencia, el primer esquema que tiene un tipo diferenciado coincidente.

Los tipos diferenciados que tienen como fuente tipos LONG VARCHAR, LONG VARGRAPHIC, LOB o DATALINK están sujetos a las mismas restricciones que el tipo fuente correspondiente. Sin embargo, se pueden especificar explícitamente determinadas funciones y operadores del tipo fuente para que se apliquen al tipo diferenciado, mediante la definición de funciones definidas por el usuario. (Estas funciones tienen como fuente las funciones definidas sobre el tipo fuente del tipo diferenciado.) Los operadores de comparación se generan automáticamente para los tipos diferenciados definidos por el usuario, a excepción de aquéllos que utilizan LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB, DBCLOB o DATALINK como tipo fuente. Además, se generar funciones para soportar una conversión del tipo de datos del tipo fuente al tipo diferenciado, y viceversa.

Conceptos relacionados:

- "Tipificación estricta en tipos diferenciados definidos por el usuario" en la página 249
- "Tipos definidos por el usuario" en la página 247

Tareas relacionadas:

- "Creación de tipos diferenciados" en la página 250
- "Creación de tablas con columnas basadas en tipos diferenciados" en la página 251
- "Manipulación de tipos diferenciados" en la página 256

Ejemplos relacionados:

- "dtudt.out -- HOW TO CREATE/USE/DROP UDTs (C)"
- "dtudt.sqc -- How to create, use, and drop user-defined distinct types (C)"
- "dtudt.out -- HOW TO CREATE/USE/DROP UDTs (C++)"
- "dtudt.sqC -- How to create, use, and drop user-defined distinct types (C++)"
- "DtUdt.java -- How to create, use and drop user defined distinct types (JDBC)"
- "DtUdt.out -- HOW TO CREATE, USE AND DROP USER DEFINED DISTINCT TYPES (JDBC)"
- "DtUdt.out -- HOW TO CREATE, USE AND DROP USER DEFINED DISTINCT TYPES (SQLJ)"
- "DtUdt.sqlj -- How to create, use and drop user defined distinct types (SQLj)"

Tipificación estricta en tipos diferenciados definidos por el usuario

Uno de los conceptos más importantes asociados con los tipos diferenciados es la tipificación estricta. La tipificación estricta garantiza que sólo las funciones y operadores definidos explícitamente en el tipo diferenciado se pueden aplicar a sus instancias.

La tipificación estricta es importante para asegurar que las instancias de los tipos diferenciados son correctas. Por ejemplo, si ha definido una función para convertir dólares americanos en dólares canadienses de acuerdo con la tasa de cambio actual, no desea que se utilice la misma función para convertir euros en dólares canadienses, porque esto devolvería una cantidad errónea.

Como consecuencia de la tipificación estricta, DB2® no le permite escribir consultas que comparan, por ejemplo, instancias de un tipo diferenciado con instancias del tipo fuente del tipo diferenciado. Por la misma razón, DB2 no le dejará aplicar funciones definidas en otros tipos a tipos diferenciados. Si desea comparar instancias de tipos diferenciados con instancias de otro tipo, tiene que convertir el tipo de datos de las instancias de uno o del otro tipo. En el mismo sentido, tiene que convertir el tipo de datos de la instancia del tipo diferenciado al tipo del parámetro de una función que no esté definida en un tipo diferenciado si desea aplicar esta función a una instancia del tipo diferenciado.

Conceptos relacionados:

- "Tipos diferenciados definidos por el usuario" en la página 247

Tareas relacionadas:

- "Creación de tipos diferenciados" en la página 250

- "Creación de tablas con columnas basadas en tipos diferenciados" en la página 251

Creación de tipos diferenciados

Un tipo diferenciado definido por el usuario es un tipo de datos deducido de un tipo existente, como por ejemplo un entero, un decimal o un tipo carácter. Cuando se crean tipos diferenciados, DB2 genera funciones de conversión del tipo de datos para convertir el tipo diferenciado al tipo fuente, y para convertir el tipo fuente al tipo diferenciado. Estas funciones resultan esenciales para la manipulación de los tipos diferenciados en consultas.

Las distintas instancias del mismo tipo diferenciado se pueden comparar entre sí, si se especifica la cláusula WITH COMPARISONS en la sentencia CREATE DISTINCT TYPE (tal como en el ejemplo del procedimiento). La cláusula WITH COMPARISONS no se puede especificar si el tipo de datos fuente es un tipo de objeto grande, DATALINK, LONG VARCHAR o LONG VARGRAPHIC.

Requisitos previos:

Para ver la lista de privilegios necesarios para definir tipos diferenciados, consulte la sentencia CREATE DISTINCT TYPE.

Restricciones:

El tipo fuente del tipo diferenciado es el tipo de datos que utiliza DB2 para representar internamente el tipo diferenciado. Por esta razón, debe ser un tipo de datos integrado. Los tipos diferenciados definidos anteriormente no se pueden utilizar como tipos fuente de otros tipos diferenciados.

Procedimiento:

Para definir un tipo diferenciado, emita la sentencia CREATE DISTINCT TYPE, especificando un nombre de tipo y el tipo fuente. Por ejemplo, la sentencia siguiente define un nuevo tipo diferenciado llamado new_type, que contiene valores SMALLINT:

```
CREATE DISTINCT TYPE new_type AS SMALLINT WITH COMPARISONS
```

Puesto que el tipo diferenciado definido en la sentencia anterior se basa en SMALLINT, se deben especificar los parámetros WITH COMPARISONS.

Para comprender mejor las aplicaciones de los tipos diferenciados definidos por el usuario, vea los ejemplos siguientes de definiciones de tipos diferenciados basados en casos comerciales de ejemplo:

- Definir tipos diferenciados basados en la moneda.
- Definir un tipo diferenciado para solicitudes de trabajo.

Conceptos relacionados:

- "Tipificación estricta en tipos diferenciados definidos por el usuario" en la página 249
- "Tipos definidos por el usuario" en la página 247
- "Tipos diferenciados definidos por el usuario" en la página 247

Tareas relacionadas:

- "Creación de tipos diferenciados basados en la moneda" en la página 253
- "Creación de un tipo diferenciado para formularios de solicitud de trabajo cumplimentados" en la página 254
- "Manipulación de tipos diferenciados" en la página 256

Información relacionada:

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

Ejemplos relacionados:

- "dtudt.out -- HOW TO CREATE/USE/DROP UDTs (C)"
- "dtudt.sqc -- How to create, use, and drop user-defined distinct types (C)"
- "dtudt.out -- HOW TO CREATE/USE/DROP UDTs (C++)"
- "dtudt.sqC -- How to create, use, and drop user-defined distinct types (C++)"
- "DtUdt.java -- How to create, use and drop user defined distinct types (JDBC)"
- "DtUdt.out -- HOW TO CREATE, USE AND DROP USER DEFINED DISTINCT TYPES (JDBC)"
- "DtUdt.out -- HOW TO CREATE, USE AND DROP USER DEFINED DISTINCT TYPES (SQLJ)"
- "DtUdt.sqlj -- How to create, use and drop user defined distinct types (SQLj)"

Creación de tablas con columnas basadas en tipos diferenciados

Después de definir tipos diferenciados, puede empezar a crear tablas con columnas basadas en tipos diferenciados.

Requisitos previos:

Para ver la lista de privilegios necesarios para definir tipos diferenciados, consulte la sentencia CREATE DISTINCT TYPE.

Para ver la lista de privilegios necesarios para crear tablas, consulte la sentencia CREATE TABLE.

Procedimiento:

Para crear una tabla con columnas basadas en tipos diferenciados:

1. Defina un tipo diferenciado:

```
CREATE DISTINCT TYPE t_educ AS SMALLINT WITH COMPARISONS
```

2. Cree la tabla, mencionado el tipo diferenciado, T_EDUC como tipo de una columna.

```
CREATE TABLE employee
  (empno CHAR(6) NOT NULL,
   firstnme VARCHAR(12) NOT NULL,
   lastname VARCHAR(15) NOT NULL,
   workdept CHAR(3),
   phoneno CHAR(4),
   photo BLOB(10M) NOT NULL,
   edlevel T_EDUC)
  IN RESOURCE
```

Para comprender mejor las aplicaciones de las tablas, vea los ejemplos siguientes de creación de tablas basadas en casos comerciales de ejemplo:

- Crear tablas para hacer un seguimiento de las ventas internacionales.
- Crear una tabla para almacenar formularios de solicitud de trabajo cumplimentados.

Conceptos relacionados:

- "Tipificación estricta en tipos diferenciados definidos por el usuario" en la página 249
- "Tipos definidos por el usuario" en la página 247
- "Tipos diferenciados definidos por el usuario" en la página 247

Tareas relacionadas:

- "Creación de tablas para hacer un seguimiento de las ventas internacionales" en la página 255
- "Creación de una tabla para almacenar formularios de solicitud de trabajo cumplimentados" en la página 255
- "Creación de tipos diferenciados" en la página 250
- "Manipulación de tipos diferenciados" en la página 256
- "Creación de tipos diferenciados basados en la moneda" en la página 253
- "Creación de un tipo diferenciado para formularios de solicitud de trabajo cumplimentados" en la página 254

Información relacionada:

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

Eliminación de tipos definidos por el usuario

Puede eliminar un tipo definido por el usuario (UDT) mediante la sentencia DROP. No se puede eliminar un UDT si éste se utiliza:

- En la definición de una columna para una tabla o vista existentes.
- Como tipo de una tabla con tipo o de una vista con tipo existentes.
- Como supertipo de otro tipo estructurado.

El gestor de bases de datos intenta eliminar cada rutina que depende de este UDT. No se puede eliminar una rutina si dependen de ella una vista, un activador, una restricción de comprobación de tablas u otra rutina. Si DB2 no puede eliminar una rutina dependiente, DB2 no elimina el UDT. La eliminación de un UDT invalida los posibles paquetes o sentencias de SQL dinámico en antememoria que lo utilizaran.

Si ha creado una transformación para un UDT y piensa eliminar dicho UDT, considere la posibilidad de eliminar la transformación asociada. Para eliminar una transformación, emita una sentencia DROP TRANSFORM. Observe que sólo puede eliminar las transformaciones definidas por el usuario. No puede eliminar las transformaciones incorporadas o las definiciones de grupos asociadas a las mismas.

Conceptos relacionados:

- "Tipos definidos por el usuario" en la página 247
- "Tipos diferenciados definidos por el usuario" en la página 247
- "Tipos estructurados definidos por el usuario" en la página 266

- "Funciones de transformación y grupos de transformación" en la página 306

Tareas relacionadas:
- "Creación de tipos diferenciados" en la página 250
- "Creación de tipos estructurados" en la página 266

Información relacionada:
- "Sentencia DROP" en la publicación *Consulta de SQL, Volumen 2*

Ejemplos relacionados:
- "dtstruct.out -- Sample C++ program : dtstruct.sqC (C++)"
- "dtstruct.sqC -- Create, use, drop a hierarchy of structured types and typed tables (C++)"
- "dtudt.out -- HOW TO CREATE/USE/DROP UDTs (C++)"
- "dtudt.sqC -- How to create, use, and drop user-defined distinct types (C++)"
- "dtudt.out -- HOW TO CREATE/USE/DROP UDTs (C)"
- "dtudt.sqc -- How to create, use, and drop user-defined distinct types (C)"
- "DtUdt.java -- How to create, use and drop user defined distinct types (JDBC)"
- "DtUdt.out -- HOW TO CREATE, USE AND DROP USER DEFINED DISTINCT TYPES (JDBC)"
- "DtUdt.out -- HOW TO CREATE, USE AND DROP USER DEFINED DISTINCT TYPES (SQLJ)"
- "DtUdt.sqlj -- How to create, use and drop user defined distinct types (SQLj)"

Creación de tipos diferenciados basados en la moneda

Supongamos que está escribiendo aplicaciones que tienen necesidad de manejar distintas monedas. Puesto que es necesario realizar conversiones cuando se desea comparar valores de distintas monedas, desea asegurarse de que DB2 no permite que se comparen ni manipulen directamente estas monedas entre sí. Dado que los tipos diferenciados sólo son compatibles consigo mismos, debe definir uno para cada moneda que tenga que representar.

Requisitos previos:

Para ver la lista de privilegios necesarios para definir tipos diferenciados, consulte la sentencia CREATE DISTINCT TYPE.

Procedimiento:

Para definir tipos diferenciados que representen el euro y las monedas americana y canadiense, emita las sentencias siguientes:

```
CREATE DISTINCT TYPE US_DOLLAR AS DECIMAL (9,3) WITH COMPARISONS
CREATE DISTINCT TYPE CANADIAN_DOLLAR AS DECIMAL (9,3) WITH COMPARISONS
CREATE DISTINCT TYPE EURO AS DECIMAL (9,3) WITH COMPARISONS
```

Observe que tiene que especificar la cláusula WITH COMPARISONS, puesto que los operadores de comparación se soportan en DECIMAL (9,3).

Conceptos relacionados:
- "Tipos diferenciados definidos por el usuario" en la página 247

Tareas relacionadas:

- "Creación de tablas con columnas basadas en tipos diferenciados" en la página 251
- "Creación de un tipo diferenciado para formularios de solicitud de trabajo cumplimentados" en la página 254
- "Creación de tablas para hacer un seguimiento de las ventas internacionales" en la página 255

Información relacionada:

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

Creación de un tipo diferenciado para formularios de solicitud de trabajo cumplimentados

Supongamos que desea conservar los formularios entrantes de solicitud de trabajo en una tabla de DB2 y poder utilizar funciones para extraer la información de dichos formularios. Puede definir un tipo diferenciado para representar a los formularios de las tablas como parámetros de las funciones.

Requisitos previos:

Para ver la lista de privilegios necesarios para definir tipos diferenciados, consulte la sentencia CREATE DISTINCT TYPE.

Procedimiento:

Para definir un tipo diferenciado que represente los formularios de solicitud de trabajo cumplimentados, emita la sentencia siguiente:

```
CREATE DISTINCT TYPE PERSONNEL.APPLICATION_FORM AS CLOB(32K)
```

Puesto que DB2 no soporta las comparaciones en CLOB, no puede especificar la cláusula WITH COMPARISONS. El esquema PERSONNEL se especifica en la sentencia anterior porque se pretende que este esquema contenga todos los tipos diferenciados y UDF que traten formularios de solicitud.

Conceptos relacionados:

- "Tipos diferenciados definidos por el usuario" en la página 247

Tareas relacionadas:

- "Creación de tablas con columnas basadas en tipos diferenciados" en la página 251
- "Creación de tipos diferenciados basados en la moneda" en la página 253
- "Creación de una tabla para almacenar formularios de solicitud de trabajo cumplimentados" en la página 255

Creación de tablas para hacer un seguimiento de las ventas internacionales

Supongamos que desea definir tablas para hacer un seguimiento de las ventas de la empresa en distintas regiones. Puede crear tablas utilizando el tipo diferenciado aplicable a la moneda como tipo de columna para los ingresos por ventas totales correspondientes a una región determinada.

Requisitos previos:

Para ver la lista de privilegios necesarios para crear tablas, consulte la sentencia CREATE TABLE.

Procedimiento:

Para crear tablas con el fin de hacer un seguimiento de las ventas internacionales:

1. Cree tipos diferenciados basados en la moneda.
2. Emita las sentencias CREATE TABLE siguientes:

```
CREATE TABLE US_SALES
  (PRODUCT_ITEM  INTEGER,
   MONTH         INTEGER CHECK (MONTH BETWEEN 1 AND 12),
   YEAR          INTEGER CHECK (YEAR > 1985),
   TOTAL         US_DOLLAR)

CREATE TABLE CANADIAN_SALES
  (PRODUCT_ITEM  INTEGER,
   MONTH         INTEGER CHECK (MONTH BETWEEN 1 AND 12),
   YEAR          INTEGER CHECK (YEAR > 1985),
   TOTAL         CANADIAN_DOLLAR)

CREATE TABLE GERMAN_SALES
  (PRODUCT_ITEM  INTEGER,
   MONTH         INTEGER CHECK (MONTH BETWEEN 1 AND 12),
   YEAR          INTEGER CHECK (YEAR > 1985),
   TOTAL         EURO)
```

Conceptos relacionados:

- "Tipos diferenciados definidos por el usuario" en la página 247

Tareas relacionadas:

- "Creación de tipos diferenciados" en la página 250
- "Creación de tipos diferenciados basados en la moneda" en la página 253
- "Creación de una tabla para almacenar formularios de solicitud de trabajo cumplimentados" en la página 255

Creación de una tabla para almacenar formularios de solicitud de trabajo cumplimentados

Supongamos que tiene que definir una tabla en la que conservar los formularios cumplimentados por solicitantes. Puede crear una tabla utilizando el tipo diferenciado PERSONNEL.APPLICATION_FORM como tipo de columna, para que contenga los formularios cumplimentados.

Requisitos previos:

Para ver la lista de privilegios necesarios para crear tablas, consulte la sentencia CREATE TABLE.

Procedimiento:

Para crear una tabla que contenga formularios de solicitud de trabajo cumplimentados:

1. Cree un tipo diferenciado para un formulario de solicitud de trabajo.
2. Emita la sentencia CREATE TABLE siguientes:

```
CREATE TABLE APPLICATIONS
  (ID               SYSIBM.INTEGER,
   NAME             VARCHAR (30),
   APPLICATION_DATE  SYSIBM.DATE,
   FORM             PERSONNEL.APPLICATION_FORM)
```

El nombre del tipo diferenciado está completamente calificado porque su calificador no es igual al ID de autorización y no ha cambiado la vía de acceso de la función por omisión. Recuerde que cuando los nombres de tipo y función no están calificados al completo, DB2 busca a través de los esquemas listados en la vía de acceso de la función actual y busca un nombre de tipo o función que coincida con el nombre no calificado especificado. Puesto que SYSIBM siempre se considera (si se ha omitido) en la vía de acceso de la función actual, puede omitir la calificación de los tipos de datos integrados.

Conceptos relacionados:

- "Tipos diferenciados definidos por el usuario" en la página 247

Tareas relacionadas:

- "Creación de tipos diferenciados" en la página 250
- "Creación de un tipo diferenciado para formularios de solicitud de trabajo cumplimentados" en la página 254
- "Creación de tablas para hacer un seguimiento de las ventas internacionales" en la página 255

Manipulación de tipos diferenciados

Manipulación de tipos diferenciados

Una vez que haya definido tipos diferenciados y creado tablas basadas en ellos, puede empezar a manipular los valores con tipo diferenciado reales.

Procedimiento:

Para implementar diversos tipos de manipulación de tipos diferenciados:

- Convertir el tipo de datos entre tipos diferenciados.
- Realizar comparaciones entre tipos diferenciados.
- Realizar comparaciones entre tipos diferenciados y constantes.
- Definir UDF de origen para tipos diferenciados.
- Realizar asignaciones que implican a tipos diferenciados.
- Realizar asignaciones que implican a tipos diferenciados en SQL dinámico.
- Realizar asignaciones que implican a distintos tipos diferenciados.
- Realizar operaciones UNION sobre columnas con tipo diferenciado.

Conceptos relacionados:

- "Tipos diferenciados definidos por el usuario" en la página 247

Tareas relacionadas:

- "Conversión del tipo de datos entre tipos diferenciados" en la página 257
- "Realización de comparaciones que implican a tipos diferenciados" en la página 258
- "Realización de comparaciones entre tipos diferenciados y constantes" en la página 259
- "Definición de UDF con fuente para tipos diferenciados" en la página 262
- "Realización de asignaciones que implican a tipos diferenciados en SQL incorporado" en la página 260
- "Realización de asignaciones que implican a tipos diferenciados en SQL dinámico" en la página 260
- "Realización de asignaciones que implican a distintos tipos diferenciados" en la página 261
- "Realización de operaciones UNION sobre columnas con tipo diferenciado" en la página 262
- "Creación de tipos diferenciados" en la página 250
- "Creación de tablas con columnas basadas en tipos diferenciados" en la página 251

Ejemplos relacionados:

- "dtudt.c -- How to create, use, and drop user-defined distinct types."
- "dtudt.out -- HOW TO CREATE/USE/DROP UDTs (C)"
- "dtudt.sqc -- How to create, use, and drop user-defined distinct types (C)"
- "dtudt.out -- HOW TO CREATE/USE/DROP UDTs (C++)"
- "dtudt.sqC -- How to create, use, and drop user-defined distinct types (C++)"
- "DtUdt.java -- How to create, use and drop user defined distinct types (JDBC)"
- "DtUdt.out -- HOW TO CREATE, USE AND DROP USER DEFINED DISTINCT TYPES (JDBC)"
- "DtUdt.out -- HOW TO CREATE, USE AND DROP USER DEFINED DISTINCT TYPES (SQLJ)"
- "DtUdt.sqlj -- How to create, use and drop user defined distinct types (SQLj)"

Conversión del tipo de datos entre tipos diferenciados

Supongamos que desea definir una UDF que convierta otra moneda en dólares americanos. A efectos de este ejemplo, puede obtener la tasa de cambio actual de una tabla tal como la siguiente:

```
CREATE TABLE
  exchange_rates(source CHAR(3), target CHAR(3), rate DECIMAL(9,3))
```

Se puede utilizar la función siguiente para acceder directamente a los valores de la tabla exchange_rates:

```
CREATE FUNCTION exchange_rate(src VARCHAR(3), trg VARCHAR(3))
  RETURNS DECIMAL(9,3)
  RETURN SELECT rate FROM exchange_rates
    WHERE source = src AND target = trg
```

Las tasas de cambio de moneda de la función anterior se basan en el tipo DECIMAL, y no en tipos diferenciados. Para representar algunas monedas distintas, utilice las definiciones de tipos diferenciados siguientes:

```
CREATE DISTINCT TYPE CANADIAN_DOLLAR AS DECIMAL (9,3) WITH COMPARISONS
CREATE DISTINCT TYPE EURO AS DECIMAL(9,3) WITH COMPARISONS
CREATE DISTINCT TYPE US_DOLLAR AS DECIMAL (9,3) WITH COMPARISONS
```

Para crear una UDF que convierta CANADIAN_DOLLAR o EURO en US_DOLLAR, deberá convertir el tipo de datos de los valores implicados. Observe que la función exchange_rate devuelve una tasa de cambio como un DECIMAL. Por ejemplo, una función que convierte valores de CANADIAN_DOLLAR en US_DOLLAR lleva a cabo los pasos siguientes:

- convertir el tipo de datos del valor CANADIAN_DOLLAR a DECIMAL
- obtener la tasa de cambio para convertir el dólar canadiense al dólar americano de la función exchange_rate, que devuelve la tasa de cambio como un valor DECIMAL
- multiplicar el valor DECIMAL del dólar canadiense por la tasa de cambio DECIMAL
- convertir el tipo de datos del valor DECIMAL a US_DOLLAR
- devolver el valor en US_DOLLAR

A continuación se muestran instancias de la función US_DOLLAR (tanto para el dólar canadiense como para el euro), que siguen los pasos anteriores.

```
CREATE FUNCTION US_DOLLAR(amount CANADIAN_DOLLAR)
  RETURNS US_DOLLAR
  RETURN US_DOLLAR(DECIMAL(amount) * exchange_rate('CAN', 'USD'))

CREATE FUNCTION US_DOLLAR(amount EURO)
  RETURNS US_DOLLAR
  RETURN US_DOLLAR(DECIMAL(amount) * exchange_rate('EUR', 'USD'))
```

Conceptos relacionados:

- "Tipos diferenciados definidos por el usuario" en la página 247

Tareas relacionadas:

- "Creación de tipos diferenciados" en la página 250
- "Creación de tablas con columnas basadas en tipos diferenciados" en la página 251
- "Definición de UDF con fuente para tipos diferenciados" en la página 262

Realización de comparaciones que implican a tipos diferenciados

Supongamos que desea saber qué productos se han vendido más en Estados Unidos que en Canadá y Alemania durante el mes de julio de 1999 (7/1999):

```
SELECT US.PRODUCT_ITEM, US.TOTAL
  FROM US_SALES AS US, CANADIAN_SALES AS CDN, GERMAN_SALES AS GERMAN
  WHERE US.PRODUCT_ITEM = CDN.PRODUCT_ITEM
  AND US.PRODUCT_ITEM = GERMAN.PRODUCT_ITEM
  AND US.TOTAL > US_DOLLAR (CDN.TOTAL)
  AND US.TOTAL > US_DOLLAR (GERMAN.TOTAL)
  AND US.MONTH = 7
  AND US.YEAR  = 1999
```

```
AND CDN.MONTH = 7
AND CDN.YEAR  = 1999
AND GERMAN.MONTH = 7
AND GERMAN.YEAR  = 1999
```

Puesto que no puede comparar directamente dólares americanos con dólares canadienses o euros, utilice la UDF para convertir el tipo de datos de la cantidad en dólares canadienses a dólares americanos, y la UDF para convertir el tipo de datos de la cantidad en euros a dólares americanos. No puede convertir el tipo de datos de todas las cantidades a DECIMAL y comparar los valores DECIMAL convertidos porque las cantidades no se pueden comparar monetariamente. Es decir, las cantidades no están en la misma moneda.

Conceptos relacionados:

- "Tipos diferenciados definidos por el usuario" en la página 247

Tareas relacionadas:

- "Creación de tipos diferenciados" en la página 250
- "Creación de tablas con columnas basadas en tipos diferenciados" en la página 251
- "Conversión del tipo de datos entre tipos diferenciados" en la página 257

Realización de comparaciones entre tipos diferenciados y constantes

Supongamos que desea saber qué productos se han vendido por valor de más de 100 000.00 dólares americanos en Estados Unidos en el mes de julio de 1999 (7/99).

```
SELECT PRODUCT_ITEM
  FROM   US_SALES
  WHERE  TOTAL > US_DOLLAR (100000)
  AND    month = 7
  AND    year  = 1999
```

Puesto que no puede comparar directamente dólares americanos con instancias del tipo fuente de dólares americanos (es decir, DECIMAL), ha utilizado la función de conversión del tipo de datos que proporciona DB2 para convertir el tipo de datos de DECIMAL a dólares americanos. También puede utilizar la otra función de conversión del tipo de datos que proporciona DB2 (es decir, la que sirve para convertir el tipo de datos de dólares americanos a DECIMAL) y convertir el tipo de datos total de la columna a DECIMAL. Cualquiera que sea su selección de difusión, del tipo diferenciado o al tipo diferenciado, puede utilizar la notación de especificación de difusión para realizar la difusión o la notación funcional. Es decir, podría haber escrito la consulta anterior del siguiente modo:

```
SELECT PRODUCT_ITEM
  FROM   US_SALES
  WHERE  TOTAL > CAST (100000 AS us_dollar)
  AND    MONTH = 7
  AND    YEAR  = 1999
```

Conceptos relacionados:

- "Tipos diferenciados definidos por el usuario" en la página 247

Tareas relacionadas:

- "Creación de tipos diferenciados" en la página 250

- "Creación de tablas con columnas basadas en tipos diferenciados" en la página 251
- "Conversión del tipo de datos entre tipos diferenciados" en la página 257

Realización de asignaciones que implican a tipos diferenciados en SQL incorporado

Supongamos que desea almacenar en la base de datos el formulario de solicitud de trabajo cumplimentado por un nuevo solicitante. Puede definir una variable del lenguaje principal que contenga el valor de serie de caracteres que se utiliza para representar el formulario cumplimentado:

```
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS CLOB(32K) hv_form;
EXEC SQL END DECLARE SECTION;

/* Código para rellenar hv_form */

INSERT INTO APPLICATIONS
  VALUES (134523, 'Peter Holland', CURRENT DATE, :hv_form)
```

No invoca de forma explícita la función de conversión del tipo de datos para convertir la serie de caracteres al tipo diferenciado `personal.application_form` porque DB2 le permite asignar instancias del tipo fuente de un tipo diferenciado a destinos que tienen dicho tipo diferenciado.

Conceptos relacionados:

- "Tipos diferenciados definidos por el usuario" en la página 247

Tareas relacionadas:

- "Creación de tipos diferenciados" en la página 250
- "Creación de tablas con columnas basadas en tipos diferenciados" en la página 251
- "Definición de UDF con fuente para tipos diferenciados" en la página 262

Realización de asignaciones que implican a tipos diferenciados en SQL dinámico

Supongamos que desea almacenar en la base de datos el formulario de solicitud de trabajo cumplimentado por un nuevo solicitante. Ha definido una variable del lenguaje principal que contiene el valor de serie de caracteres que se utiliza para representar el formulario cumplimentado. Para utilizar SQL dinámico puede usar marcadores de parámetros, del modo siguiente:

```
EXEC SQL BEGIN DECLARE SECTION;
  long id;
  char name[30];
  SQL TYPE IS CLOB(32K) form;
  char command[80];
EXEC SQL END DECLARE SECTION;

/* Código para rellenar variables del lenguaje principal */

strcpy(command,"INSERT INTO APPLICATIONS VALUES");
strcat(command,"(?, ?, CURRENT DATE, CAST (? AS CLOB(32K)))");

EXEC SQL PREPARE APP_INSERT FROM :command;
EXEC SQL EXECUTE APP_INSERT USING :id, :name, :form;
```

Esta operación hace uso de la especificación de conversión del tipo de datos de DB2 para indicar a DB2 que el tipo de marcador de parámetro es CLOB(32K), un tipo que se puede asignar a la columna de tipo diferenciado. Recuerde que no puede declarar una variable del lenguaje principal de un tipo diferenciado, puesto que los lenguajes principales no soportan tipos diferenciados. Por lo tanto, no puede especificar que el tipo de un marcador de parámetro es un tipo diferenciado.

Conceptos relacionados:

- "Tipos diferenciados definidos por el usuario" en la página 247

Tareas relacionadas:

- "Creación de tipos diferenciados" en la página 250
- "Creación de tablas con columnas basadas en tipos diferenciados" en la página 251

Realización de asignaciones que implican a distintos tipos diferenciados

Supongamos que ha definido una UDF fuente en la función incorporada SUM para dar soporte a SUM en dólares americanos y canadienses:

```
CREATE FUNCTION SUM (CANADIAN_DOLLAR)
  RETURNS CANADIAN_DOLLAR
  SOURCE SYSIBM.SUM (DECIMAL())

CREATE FUNCTION SUM (US_DOLLAR)
  RETURNS US_DOLLAR
  SOURCE SYSIBM.SUM (DECIMAL())
```

Supongamos ahora que el supervisor le pide que mantenga las ventas totales anuales en dólares americanos de cada producto y cada región, en tablas separadas:

```
CREATE TABLE US_SALES_94
  (PRODUCT_ITEM  INTEGER,
   TOTAL         US_DOLLAR)

CREATE TABLE GERMAN_SALES_94
  (PRODUCT_ITEM  INTEGER,
   TOTAL         US_DOLLAR)

CREATE TABLE CANADIAN_SALES_94
  (PRODUCT_ITEM  INTEGER,
   TOTAL         US_DOLLAR)

INSERT INTO US_SALES_94
  SELECT PRODUCT_ITEM, SUM (TOTAL)
  FROM US_SALES
  WHERE YEAR = 1994
  GROUP BY PRODUCT_ITEM

INSERT INTO GERMAN_SALES_94
  SELECT PRODUCT_ITEM, US_DOLLAR (SUM (TOTAL))
  FROM GERMAN_SALES
  WHERE YEAR = 1994
  GROUP BY PRODUCT_ITEM

INSERT INTO CANADIAN_SALES_94
```

```
SELECT PRODUCT_ITEM, US_DOLLAR (SUM (TOTAL))
FROM CANADIAN_SALES
WHERE YEAR = 1994
GROUP BY PRODUCT_ITEM
```

Convierte de forma explícita las cantidades en dólares canadienses y euros a dólares americanos, puesto que los distintos tipos diferenciados no se pueden asignar directamente entre sí. No puede utilizar la sintaxis de la especificación de difusión porque el tipo de datos de los tipos diferenciados sólo se puede convertir a su propio tipo fuente.

Conceptos relacionados:

- "Tipos diferenciados definidos por el usuario" en la página 247

Tareas relacionadas:

- "Creación de tipos diferenciados" en la página 250
- "Creación de tablas con columnas basadas en tipos diferenciados" en la página 251

Realización de operaciones UNION sobre columnas con tipo diferenciado

Supongamos que desea proporcionar a los usuarios americanos una vista que contenga todas las ventas de cada producto de la empresa:

```
CREATE VIEW ALL_SALES AS
  SELECT PRODUCT_ITEM, MONTH, YEAR, TOTAL
  FROM US_SALES
  UNION
  SELECT PRODUCT_ITEM, MONTH, YEAR, US_DOLLAR (TOTAL)
  FROM CANADIAN_SALES
  UNION
  SELECT PRODUCT_ITEM, MONTH, YEAR, US_DOLLAR (TOTAL)
  FROM GERMAN_SALES
```

Convierte el tipo de datos de los dólares canadienses a dólares americanos y euros a dólares americanos porque los tipos diferenciados sólo son compatibles para la función de unión con el mismo tipo diferenciado. El ejemplo anterior hace uso de las UDF definidas en Difusión entre tipos diferenciados para convertir el tipo de datos entre las monedas, lo cual tiene como consecuencia la utilización de la notación funcional en lugar de una especificación de conversión del tipo de datos.

Conceptos relacionados:

- "Tipos diferenciados definidos por el usuario" en la página 247

Tareas relacionadas:

- "Creación de tipos diferenciados" en la página 250
- "Creación de tablas con columnas basadas en tipos diferenciados" en la página 251
- "Conversión del tipo de datos entre tipos diferenciados" en la página 257

Definición de UDF con fuente para tipos diferenciados

Supongamos que ha definido una UDF fuente en la función SUM integrada para dar soporte a SUM en euros:

```
CREATE FUNCTION SUM (EUROS)
  RETURNS EUROS
  SOURCE SYSIBM.SUM (DECIMAL())
```

Desea saber el total de ventas en Alemania correspondiente a cada producto durante el año 1994. Desea obtener las ventas totales en dólares americanos:

```
SELECT PRODUCT_ITEM, US_DOLLAR (SUM (TOTAL))
  FROM GERMAN_SALES
  WHERE YEAR = 1994
  GROUP BY PRODUCT_ITEM
```

No podría escribir `SUM (us_dollar (total))`, a no ser que hubiera definido una función SUM en dólares americanos de forma parecida a la anterior.

Conceptos relacionados:

- "Tipos diferenciados definidos por el usuario" en la página 247

Tareas relacionadas:

- "Creación de tipos diferenciados" en la página 250
- "Creación de tablas con columnas basadas en tipos diferenciados" en la página 251
- "Realización de asignaciones que implican a tipos diferenciados en SQL incorporado" en la página 260

Capítulo 8. Tipos estructurados definidos por el usuario

Tipos estructurados definidos por el usuario . . . 266
Creación de tipos estructurados 266
Almacenamiento de instancias de tipos estructurados 267
Posibilidad de creación de instancias en tipos estructurados 268
Jerarquías de tipos estructurados 268
Creación de una jerarquía de tipos estructurados 270
Definición del comportamiento de los tipos estructurados 271
Despacho dinámico de métodos 272
Rutinas generadas por el sistema para tipos estructurados 274
Funciones de comparación y de conversión del tipo de datos para tipos estructurados 274
Funciones de constructor para tipos estructurados 274
Métodos de mutador para tipos estructurados 275
Métodos de observador para tipos estructurados 275
Tablas con tipo 276
Tablas con tipo 276
Creación de tablas con tipo 276
Eliminación de tablas con tipo 279
Posibilidad de sustitución en las tablas con tipo 280
Almacenamiento de objetos en filas de tablas con tipo 281
Definición de identificadores de objetos generados por el sistema 283
Definición de restricciones sobre columnas de identificador de objetos 285
Tipos de referencia 286
Tipos de referencia 286
Relaciones entre objetos de las tablas con tipo 287
Definición de relaciones semánticas con referencias 288
Integridad referencial frente a referencias con ámbito 290
Vistas con tipo 290
Vistas con tipo 290
Creación de vistas con tipo 291
Modificación de vistas con tipo 293
Eliminación de vistas con tipo 293
Consulta de tablas con tipo y vistas con tipo . . . 294
Emisión de consultas para deshacer referencias 294
Devolución de objetos de un tipo determinado mediante ONLY 296
Restricción de los tipos devueltos mediante un predicado TYPE 296
Devolución de todos los tipos posibles mediante OUTER 297
Tipos estructurados como tipos de columna . . . 298
Almacenamiento de objetos de tipo estructurado en columnas de tablas 298
Inserción en columnas de atributos de tipo estructurado 300
Definición y modificación de tablas con columnas de tipo estructurado 301
Definición de tipos con atributos de tipo estructurado 301
Inserción de filas que contienen valores de tipo estructurado 302
Modificación de valores de tipo estructurado en columnas 303
Recuperación y modificación de valores de tipo estructurado en las columnas 303
Recuperación de atributos de tipo estructurado 304
Acceso a los atributos de subtipos 305
Modificación de atributos de tipo estructurado 305
Devolución de información sobre un tipo estructurado 306
Funciones de transformación y grupos de transformación 306
Funciones de transformación y grupos de transformación 306
Recomendaciones para denominar grupos de transformación 307
Especificación de grupos de transformación . . 308
Especificación de grupos de transformación 308
Especificación de grupos de transformación para rutinas externas 309
Especificación de grupos de transformación para SQL dinámico 309
Especificación de grupos de transformación para SQL estático 310
Creación de la correlación con el programa del lenguaje principal 310
Correlaciones del programa del lenguaje principal con funciones de transformación. . . 310
Transformaciones de función 311
Implementación de transformaciones de función utilizando rutinas incorporadas al SQL 313
Pase de parámetros de tipo estructurado a rutinas externas 314
Transformaciones de cliente 316
Implementación de transformaciones de cliente mediante UDF externas 319
Implementación de transformaciones de cliente para enlazar desde un cliente mediante UDF externas 319
Consideraciones sobre la conversión de datos 320
Requisitos de las funciones de transformación 321
Recuperación de datos de subtipo de DB2 . . . 322
Devolución de datos de subtipo a DB2 325
Variables del lenguaje principal de tipo estructurado 329
Declaración de variables del lenguaje principal de tipo estructurado 329
Descripción de un tipo estructurado 329

Tipos estructurados definidos por el usuario

Un tipo estructurado es un tipo de datos definido por el usuario que contiene uno o más atributos con nombre, cada uno de los cuales tiene un tipo de datos. Los atributos son propiedades que describen una instancia de un tipo. Por ejemplo, una figura geométrica puede tener atributos tales como su lista de coordenadas cartesianas. Una persona puede tener como atributos su nombre, dirección, etc. Un departamento puede tener como atributos un nombre o algún otro tipo de ID.

Un tipo estructurado incluye también un conjunto de especificaciones de métodos. Los métodos permiten definir comportamientos de los tipos estructurados. Al igual que las funciones definidas por el usuario (UDF), los métodos son rutinas que amplían el SQL. No obstante, en el caso de los métodos el comportamiento se integra únicamente en un tipo estructurado determinado.

Un tipo estructurado se puede utilizar como tipo de una tabla, vista o columna. Si se utiliza como tipo de una tabla o vista, dichas tabla o vista se conocen, respectivamente, como tabla con tipo o vista con tipo. Para las tablas con tipo y las vistas con tipo, los nombres y tipos de datos de los atributos del tipo estructurado se convierten en nombres y tipos de datos de las columnas de la tabla o vista con tipo. Las filas de la tabla o vista con tipo se pueden contemplar como una representación de instancias del tipo estructurado.

Un tipo no se puede eliminar si otros determinados objetos lo utilizan, ya sea directa o indirectamente. Por ejemplo, no se puede eliminar un tipo si una columna de una tabla o vista hace un uso directo o indirecto del tipo.

Conceptos relacionados:

- "Tipos definidos por el usuario" en la página 247
- "Tablas con tipo" en la página 276
- "Vistas con tipo" en la página 290

Tareas relacionadas:

- "Creación de tipos estructurados" en la página 266
- "Almacenamiento de instancias de tipos estructurados" en la página 267
- "Definición del comportamiento de los tipos estructurados" en la página 271
- "Eliminación de tipos definidos por el usuario" en la página 252

Ejemplos relacionados:

- "dtstruct.out -- Sample C++ program : dtstruct.sqC (C++)"
- "dtstruct.sqC -- Create, use, drop a hierarchy of structured types and typed tables (C++)"

Creación de tipos estructurados

Un tipo estructurado es un tipo definido por el usuario que contiene uno o más atributos, cada uno de los cuales tiene un nombre y un tipo de datos propios. Un tipo estructurado puede servir de tipo de una tabla o vista en que cada columna de la tabla deduce su nombre y tipo de datos de uno de los atributos del tipo estructurado. Asimismo, un tipo estructurado puede servir de tipo de una columna o de tipo de un argumento para una rutina.

Requisitos previos:

Para ver la lista de privilegios necesarios para definir tipos estructurados, consulte la sentencia CREATE TYPE.

Procedimiento:

Para definir un tipo estructurado para representar a una persona, con atributos de edad y dirección, emita la sentencia siguiente:

```
CREATE TYPE Person_t AS
   (Name VARCHAR(20),
   Age INT,
   Address Address_t)
   INSTANTIABLE
   REF USING VARCHAR(13) FOR BIT DATA
   MODE DB2SQL;
```

A diferencia de los tipos diferenciados, los atributos de los tipos estructurados se pueden componer de tipos distintos de los tipos de datos incorporados en DB2. La declaración de tipo anterior incluye un atributo denominado `Address` cuyo tipo fuente es otro tipo estructurado, `Address_t`.

Conceptos relacionados:

- "Tipos diferenciados definidos por el usuario" en la página 247
- "Tipos estructurados definidos por el usuario" en la página 266
- "Jerarquías de tipos estructurados" en la página 268

Tareas relacionadas:

- "Almacenamiento de instancias de tipos estructurados" en la página 267
- "Creación de una jerarquía de tipos estructurados" en la página 270
- "Eliminación de tipos definidos por el usuario" en la página 252

Información relacionada:

- "Sentencia CREATE TYPE (Estructurado)" en la publicación *Consulta de SQL, Volumen 2*

Ejemplos relacionados:

- "dtstruct.out -- Sample C++ program : dtstruct.sqC (C++)"
- "dtstruct.sqC -- Create, use, drop a hierarchy of structured types and typed tables (C++)"

Almacenamiento de instancias de tipos estructurados

Una instancia de un tipo estructurado se puede almacenar en la base de datos de dos maneras:

- Como fila de una tabla, en que cada columna de la tabla es un atributo de la instancia del tipo. Si necesita hacer referencia a una instancia desde otras tablas, debe utilizar tablas con tipo. Para almacenar objetos como filas de una tabla, la tabla se define con el tipo estructurado, en lugar de especificar columnas individuales en la definición de la tabla:

  ```
  CREATE TABLE Person OF Person_t
     ...
  ```

Cada columna de la tabla deduce su nombre y tipo de datos de uno de los atributos del tipo estructurado indicado. Estas tablas se conocen como tablas con tipo.

- Como valor de una columna. Para almacenar objetos en columnas de una tabla, la columna se define utilizado como tipo el tipo estructurado. La sentencia siguiente crea una tabla `Properties` que tiene un tipo estructurado `Address`, que es del tipo estructurado `Address_t`:

```
CREATE TABLE Properties
   (ParcelNum INT,
   Photo BLOB(2K),
   Address Address_t)
   ...
```

Conceptos relacionados:

- "Tipos estructurados definidos por el usuario" en la página 266
- "Tablas con tipo" en la página 276

Tareas relacionadas:

- "Almacenamiento de objetos en filas de tablas con tipo" en la página 281
- "Almacenamiento de objetos de tipo estructurado en columnas de tablas" en la página 298

Posibilidad de creación de instancias en tipos estructurados

Los tipos también se pueden definir como *INSTANTIABLE* (se pueden crear instancias) o como *NOT INSTANTIABLE* (no se pueden crear instancias). Por omisión, se pueden crear instancias de los tipos, lo cual significa que se puede crear una instancia del objeto. A la inversa, los tipos de los que no se pueden crear instancias sirven de modelos destinados a un ajuste adicional de la jerarquía de tipos. Por ejemplo, si define `Person_t` utilizando la cláusula NOT INSTANTIABLE, no podrá almacenar ninguna instancia de una persona en la base de datos ni podrá crear una tabla o vista utilizando `Person_t`. En cambio, sólo podrá almacenar instancias de `Employee_t` o de otros subtipos de `Person_t` que defina.

Conceptos relacionados:

- "Tipos estructurados definidos por el usuario" en la página 266

Tareas relacionadas:

- "Creación de tipos estructurados" en la página 266

Información relacionada:

- "Sentencia CREATE TYPE (Estructurado)" en la publicación *Consulta de SQL, Volumen 2*

Jerarquías de tipos estructurados

Ciertamente, es posible modelar objetos tales como personas utilizando tablas y columnas relacionales tradicionales. Sin embargo, los tipos estructurados ofrecen una propiedad adicional, la *herencia*. Es decir, un tipo estructurado puede tener *subtipos* que vuelvan a utilizar todos sus atributos y contengan atributos adicionales específicos del subtipo. El tipo original es el *supertipo*. Por ejemplo, el tipo estructurado `Person_t` puede contener atributos para `Name`, `Age` y `Address`. Un subtipo de `Person_t` puede ser `Employee_t`, que contiene todos los atributos `Name`,

Age y Address y, además, contiene atributos para SerialNum, Salary y BusinessUnit.

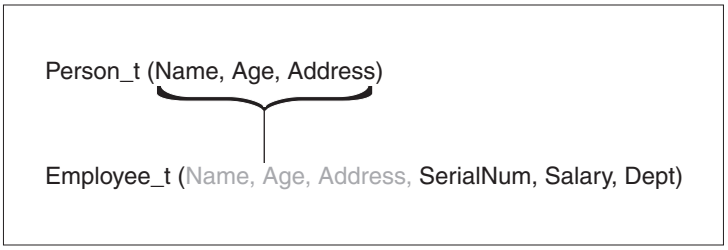


Figura 3. El tipo estructurado Employee_t hereda los atributos del supertipo Person_t

Un conjunto de subtipos basado (a cierto nivel) en el mismo supertipo se conoce como una jerarquía de tipos. Por ejemplo, es posible que un modelo de datos tenga que representar un tipo especial de empleado llamado director (manager). Los directores tienen más atributos que los empleados que no lo son. El tipo Manager_t hereda los atributos definidos para un empleado, pero también se define con algunos atributos adicionales propios, como por ejemplo un atributo de bonificación especial que sólo está disponible para los directores.

La figura siguiente presenta una ilustración de los diversos subtipos que se pueden deducir de los tipos Person y Employee:

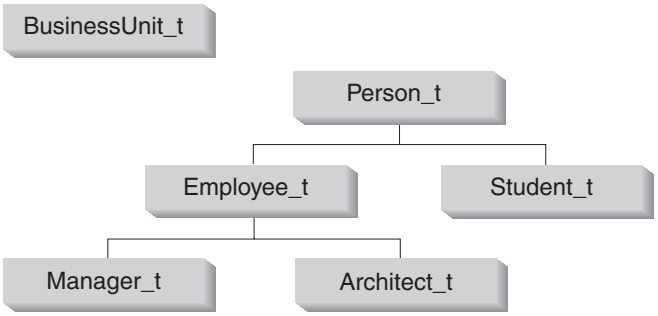


Figura 4. Jerarquías de tipos (BusinessUnit_t y Person_t)

En la Figura 4, el tipo de persona Person_t es el *tipo raíz* de la jerarquía. Person_t también es supertipo de los tipos que le siguen--en este caso, los tipos denominados Employee_t y Student_t. Las relaciones entre subtipos y supertipos son transitivas; en otras palabras, la relación entre subtipo y supertipo existe en toda la jerarquía de tipos. Así, Person_t también es supertipo de los tipos Manager_t y Architect_t.

El tipo de departamento, BusinessUnit_t se considera una jerarquía de tipos trivial. Es la raíz de una jerarquía que no tiene ningún subtipo.

Conceptos relacionados:

- "Tipos estructurados definidos por el usuario" en la página 266

Tareas relacionadas:

- "Creación de tipos estructurados" en la página 266
- "Creación de una jerarquía de tipos estructurados" en la página 270

Creación de una jerarquía de tipos estructurados

La figura siguiente presenta una ilustración de una jerarquía de tipos estructurados:

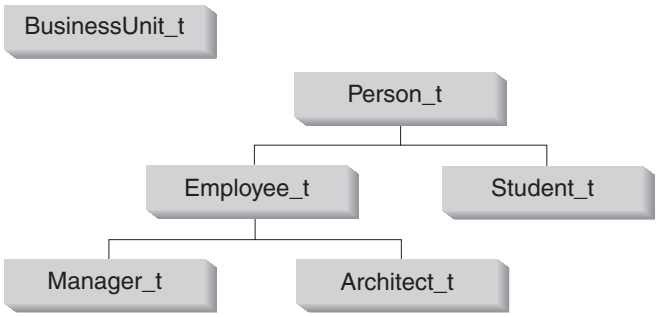


Figura 5. Jerarquías de tipos (BusinessUnit_t y Person_t)

Para crear el tipo `BusinessUnit_t`, emita la sentencia CREATE TYPE de SQL siguiente:

```
CREATE TYPE BusinessUnit_t AS
   (Name VARCHAR(20),
   Headcount INT)
   MODE DB2SQL;
```

Para crear la jerarquía de tipos `Person_t`, emita las sentencias de SQL siguientes:

```
CREATE TYPE Person_t AS
   (Name VARCHAR(20),
   Age INT,
   Address Address_t)
   REF USING VARCHAR(13) FOR BIT DATA
   MODE DB2SQL;

CREATE TYPE Employee_t UNDER Person_t AS
   (SerialNum INT,
   Salary DECIMAL (9,2),
   Dept REF(BusinessUnit_t))
   MODE DB2SQL;

CREATE TYPE Student_t UNDER Person_t AS
   (SerialNum CHAR(6),
   GPA DOUBLE)
   MODE DB2SQL;

CREATE TYPE Manager_t UNDER Employee_t AS
   (Bonus DECIMAL (7,2))
   MODE DB2SQL;

CREATE TYPE Architect_t UNDER Employee_t AS
   (StockOption INTEGER)
   MODE DB2SQL;
```

`Person_t` tiene tres atributos: `Name`, `Age` y `Address`. Sus dos subtipos, `Employee_t` y `Student_t`, heredan los atributos de `Person_t` y tienen también varios atributos adicionales que son específicos de sus tipos en concreto. Por ejemplo, aunque tanto los empleados como los estudiantes tienen números de serie, el formato utilizado para los números de serie de los estudiantes es distinto del formato utilizado para los empleados.

Finalmente, Manager_t y Architect_t son subtipos de Employee_t; heredan todos los atributos de Employee_t y los amplían más según corresponda a sus propios tipos. Así, una instancia del tipo Manager_t tendrá un total de siete atributos: Name, Age, Address, SerialNum, Salary, Dept y Bonus.

Conceptos relacionados:

- "Tipos estructurados definidos por el usuario" en la página 266
- "Jerarquías de tipos estructurados" en la página 268

Tareas relacionadas:

- "Creación de tipos estructurados" en la página 266
- "Creación de tablas con tipo" en la página 276

Información relacionada:

- "Sentencia CREATE TYPE (Estructurado)" en la publicación *Consulta de SQL, Volumen 2*

Ejemplos relacionados:

- "dtstruct.out -- Sample C++ program : dtstruct.sqC (C++)"
- "dtstruct.sqC -- Create, use, drop a hierarchy of structured types and typed tables (C++)"

Definición del comportamiento de los tipos estructurados

Para definir comportamientos de los tipos estructurados, puede crear métodos definidos por el usuario. No se pueden crear métodos para tipos diferenciados. La creación de un método es parecida a la creación de una función, con la excepción de que los métodos se crean específicamente para un tipo, de forma que el tipo y su comportamiento están estrechamente integrados.

La especificación del método se debe asociar al tipo antes de emitir la sentencia CREATE METHOD. La sentencia siguiente añade la especificación de método para un método llamado calc_bonus al tipo Employee_t:

```
ALTER TYPE Employee_t
   ADD METHOD calc_bonus (rate DOUBLE)
   RETURNS DECIMAL(7,2)
   LANGUAGE SQL
   CONTAINS SQL
   NO EXTERNAL ACTION
   DETERMINISTIC;
```

Una vez que haya asociado la especificación de método al tipo, puede definir el comportamiento del tipo creando el método, ya sea como método externo o como método en forma de SQL, según la especificación del método. Por ejemplo, la sentencia siguiente registra un método de SQL denominado calc_bonus que reside en el mismo esquema que el tipo Employee_t:

```
CREATE METHOD calc_bonus (rate DOUBLE)
   RETURNS DECIMAL(7,2)
   FOR Employee_t
   RETURN SELF..salary * rate;
```

Puede crear tantos métodos llamados calc_bonus como quiera, siempre que tengan números o tipos de parámetro distintos, o se definan para tipos que se encuentren en distintas jerarquías de tipos. En otras palabras, no se puede crear otro método

denominado `calc_bonus` para `Architect_t` que tenga los mismos tipos de parámetro y el mismo número de parámetros.

Conceptos relacionados:

- "Tipos estructurados definidos por el usuario" en la página 266
- "Despacho dinámico de métodos" en la página 272

Tareas relacionadas:

- "Creación de tipos estructurados" en la página 266

Información relacionada:

- "Sentencia ALTER TYPE (Estructurado)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE TABLE" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE METHOD" en la publicación *Consulta de SQL, Volumen 2*

Despacho dinámico de métodos

El comportamiento de un tipo estructurado se representa por sus métodos. Sólo se puede invocar estos métodos sobre instancias de su propio tipo estructurado. Cuando se crea un subtipo, entre los atributos que éste hereda se encuentran los métodos definidos para el supertipo. Por eso, los métodos de un supertipo también se pueden ejecutar sobre las instancias de sus subtipos.

Si no desea que un método definido para un supertipo se utilice para un subtipo determinado, puede alterar temporalmente el método. Alterar temporalmente un método significa volverlo a implementar específicamente para un subtipo determinado. Esto facilita el despacho dinámico de métodos (también conocido como polimorfismo), en que una aplicación ejecutará el método más específico en función del tipo de instancia del tipo estructurado (por ejemplo, en qué lugar de la jerarquía de tipos estructurados está situado).

Para definir un método de alteración temporal, utilice la sentencia CREATE TYPE (o ALTER TYPE) y especifique la cláusula OVERRIDING antes que la cláusula METHOD. Si no se especifica OVERRIDING, se utilizará el método original (que pertenece al supertipo). Para que se defina un método de alteración temporal, se tienen que cumplir las condiciones siguientes:

- El tipo que esté creando (o modificando) tiene que ser un subtipo del tipo estructurado cuyo método pretende alterar temporalmente.
- La signatura (el nombre y la lista de parámetros del método) del método que esté declarando tiene que ser idéntica a la de un método que pertenezca al supertipo.
- Un método de alteración temporal debe alterar temporal y explícitamente exactamente un método original.
- La rutina que pretenda utilizar para la alteración temporal tiene que ser un método de instancia de tipo estructurado definido por el usuario.
- El método original no se debe declarar con PARAMETER STYLE JAVA.

El ejemplo siguiente demuestra un escenario de ejemplo para la alteración temporal de métodos:

Tipos de datos:

```
CREATE TYPE a AS (z VARCHAR(20))
    METHOD foo(i INTEGER) RETURNS VARCHAR(80)
        LANGUAGE SQL;

CREATE TYPE b UNDER a AS (y VARCHAR(20))
    OVERRIDING METHOD foo(i INTEGER) RETURNS VARCHAR(80);

CREATE TYPE c UNDER a AS (x VARCHAR(20))
    OVERRIDING METHOD foo(i INTEGER) RETURNS VARCHAR(80);

CREATE TYPE d UNDER b AS (w VARCHAR(20))
    OVERRIDING METHOD foo(i INTEGER) RETURNS VARCHAR(80);
```

En esta situación, a es el supertipo. Los tipos b y c son subtipos de a. Finalmente, d es el subtipo de b

Métodos:

```
CREATE METHOD foo(i INTEGER) FOR a
  RETURN "In method foo_a. Input: " | char(i) | self..z | ".";

CREATE METHOD foo(i INTEGER) FOR b
  RETURN "In method foo_b. Input: " | char(i) | self..z |
         " y = " | self..y | ".";

CREATE METHOD foo(i INTEGER) FOR c
  RETURN "In method foo_c. Input: " | char(i) | self..z |
         " y = " | self..y | " x = " | self..x | ".";

CREATE METHOD foo(i INTEGER) FOR d
  RETURN "In method foo_d. Input: " | char(i) | self..z |
         " y = " | self..y | " w = " | self..w | ".";
```

Aquí, el método original es fooA. fooB, fooC y fooD alteran temporal y explícitamente fooA. fooD altera temporal e implícitamente fooB y fooA. De forma parecida, fooB altera temporal e implícitamente fooA, y fooC altera temporal e implícitamente fooA. (Observe que una alteración temporal explícita implica una alteración temporal implícita.)

Conceptos relacionados:

- "Tipos estructurados definidos por el usuario" en la página 266
- "Jerarquías de tipos estructurados" en la página 268

Tareas relacionadas:

- "Creación de tipos estructurados" en la página 266
- "Definición del comportamiento de los tipos estructurados" en la página 271

Información relacionada:

- "Sentencia ALTER TYPE (Estructurado)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE TYPE (Estructurado)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE METHOD" en la publicación *Consulta de SQL, Volumen 2*

Rutinas generadas por el sistema para tipos estructurados

Funciones de comparación y de conversión del tipo de datos para tipos estructurados

DB2® crea automáticamente funciones que convierten el tipo de datos de los valores entre el tipo de referencia y su tipo de representación, en ambas direcciones. La sentencia CREATE TYPE tiene una cláusula CAST WITH opcional, que permite elegir los nombres de estas dos funciones de conversión del tipo de datos. Por omisión, los nombres de dichas funciones son iguales que los nombres del tipo estructurado y de su tipo de representación de referencia. Por ejemplo, la sentencia CREATE TYPE Person_t crea automáticamente funciones con el formato siguiente:

```
CREATE FUNCTION VARCHAR(REF(Person_t))
   RETURNS VARCHAR
```

DB2 crea también la función que realiza la operación inversa:

```
CREATE FUNCTION Person_t(VARCHAR(13))
   RETURNS REF(Person_t)
```

El usuario utilizará estas funciones de conversión del tipo de datos siempre que tenga necesidad de insertar un nuevo valor en la tabla con tipo, o cuando desee comparar un valor de referencia con otro valor.

DB2 también crea funciones que permiten comparar tipos de referencia utilizando los operadores de comparación siguientes: =, <>, <, <=, > y >=.

Conceptos relacionados:

- "Tipos estructurados definidos por el usuario" en la página 266
- "Tipos de referencia" en la página 286

Tareas relacionadas:

- "Creación de tipos estructurados" en la página 266

Información relacionada:

- "Sentencia CREATE TYPE (Estructurado)" en la publicación *Consulta de SQL, Volumen 2*

Funciones de constructor para tipos estructurados

Cuando se crea un tipo estructurado, DB2® crea una función con el mismo nombre que el tipo creado. Esta función no tiene ningún parámetro y devuelve una instancia del tipo con todos los atributos del mismo establecidos como nulos. La función que se crea, por ejemplo, para el tipo estructurado `Person_t` tiene el formato siguiente:

```
CREATE FUNCTION Person_t ( ) RETURNS Person_t
```

Para el subtipo `Manager_t`, se crea un constructor con el formato siguiente:

```
CREATE FUNCTION Manager_t ( ) RETURNS Manager_t
```

Para construir una instancia de un tipo para insertarla en una columna, utilice la función de constructor con los métodos de mutador. Si el tipo se almacena en una

tabla, en lugar de hacerlo en una columna, no es necesario que utilice la función de constructor con los métodos de mutador para insertar una instancia de un tipo.

Conceptos relacionados:

- "Tipos estructurados definidos por el usuario" en la página 266

Tareas relacionadas:

- "Creación de tipos estructurados" en la página 266

Métodos de mutador para tipos estructurados

Existe un método de mutador para cada atributo de un objeto. La instancia de un tipo estructurado en la que se invoca un método se denomina instancia *sujeto* del método. Cuando el método de mutador invocado en una instancia sujeto recibe un valor nuevo para un atributo, el método devuelve una nueva instancia con el atributo actualizado con el nuevo valor. Por lo tanto, para el tipo `Person_t`, DB2® crea métodos de mutador para cada uno de los atributos siguientes: `name`, `age` y `address`.

Por ejemplo, el método de mutador que DB2 crea para el atributo `age` tiene el formato siguiente:

```
ALTER TYPE Person_t
   ADD METHOD AGE(int)
   RETURNS Person_t;
```

Conceptos relacionados:

- "Tipos estructurados definidos por el usuario" en la página 266

Tareas relacionadas:

- "Creación de tipos estructurados" en la página 266

Métodos de observador para tipos estructurados

Existe un método de observador para cada atributo de un objeto. Si el método para un atributo recibe un objeto del tipo o subtipo esperado, el método devuelve el valor del atributo para dicho objeto.

Por ejemplo, el método de observador que DB2® crea para el atributo `age` del tipo `Person_t` tiene el formato siguiente:

```
ALTER TYPE Person_t
   ADD METHOD AGE()
   RETURNS INTEGER;
```

Para invocar un método en un tipo estructurado, utilice el operador de invocación de método: '..'.

El ejemplo siguiente demuestra el uso de los métodos de observador para el tipo Person_t:

```
CREATE FUNCTION MailingAddress (p Person_t)
  RETURNS VARCHAR(40)
  RETURN p..name() || ' ' || p..address()
```

En esta función, la columna de nombre y la columna de dirección de una instancia de Person_t se recuperan a través de sus métodos de observador y se concatenan en una sola serie para formar una dirección de correo.

Conceptos relacionados:

- "Tipos estructurados definidos por el usuario" en la página 266

Tareas relacionadas:

- "Creación de tipos estructurados" en la página 266

Tablas con tipo

Tablas con tipo

Las tablas con tipo son tablas que se definen con un tipo estructurado definido por el usuario. Mediante las tablas con tipo, puede establecer una estructura jerárquica con una relación definida entre dichas tablas, denominada jerarquía de tablas. La jerarquía de tablas está formada por una sola tabla raíz, supertablas y subtablas.

Las tablas con tipo almacenan instancias de tipos estructurados como filas, en las cuales cada atributo del tipo se almacena en una columna separada.

Conceptos relacionados:

- "Tipos estructurados definidos por el usuario" en la página 266
- "Tipos de referencia" en la página 286
- "Posibilidad de sustitución en las tablas con tipo" en la página 280
- "Vistas con tipo" en la página 290

Tareas relacionadas:

- "Almacenamiento de objetos en filas de tablas con tipo" en la página 281
- "Eliminación de tablas con tipo" en la página 279
- "Definición de identificadores de objetos generados por el sistema" en la página 283
- "Definición de restricciones sobre columnas de identificador de objetos" en la página 285
- "Creación de tablas con tipo" en la página 276

Información relacionada:

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

Creación de tablas con tipo

Las tablas con tipo se utilizan para almacenar realmente instancias de objetos cuyas características se definen con la sentencia CREATE TYPE. Puede crear una tabla con tipo utilizando una variante de la sentencia CREATE TABLE. También puede crear una jerarquía de tablas con tipo que se base en una jerarquía de tipos estructurados. Para almacenar instancias de subtipos en tablas con tipo, debe crear una jerarquía de tablas correspondiente.

La figura siguiente ilustra una jerarquía de tablas con tipo. El ejemplo que sigue a la figura ilustra la creación de dicha jerarquía.

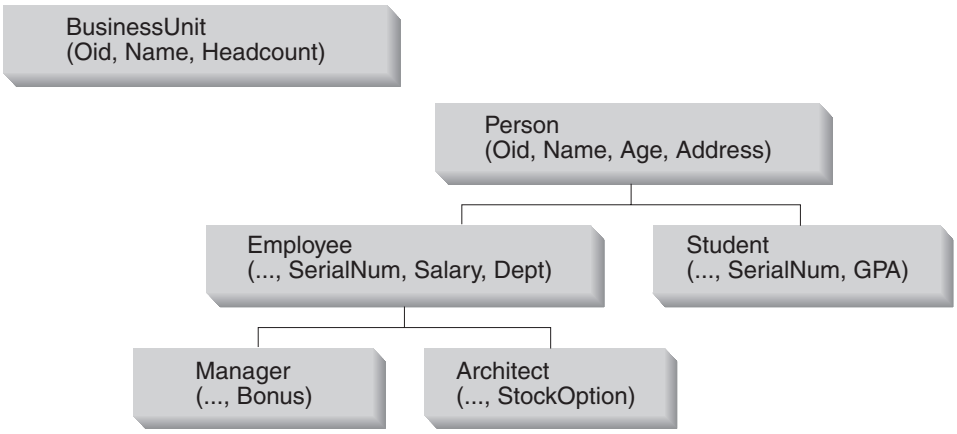


Figura 6. Jerarquía de tablas con tipo

Éste es el SQL necesario para crear la tabla con tipo `BusinessUnit`:

```
CREATE TABLE BusinessUnit OF BusinessUnit_t
(REF IS Oid USER GENERATED);
```

Éste es el SQL necesario para crear las tablas de la jerarquía de tablas `Person`:

```
CREATE TABLE Person OF Person_t
(REF IS Oid USER GENERATED);

CREATE TABLE Employee OF Employee_t UNDER Person
   INHERIT SELECT PRIVILEGES
   (SerialNum WITH OPTIONS NOT NULL,
   Dept WITH OPTIONS SCOPE BusinessUnit );

CREATE TABLE Student OF Student_t UNDER Person
   INHERIT SELECT PRIVILEGES;

CREATE TABLE Manager OF Manager_t UNDER Employee
   INHERIT SELECT PRIVILEGES;

CREATE TABLE Architect OF Architect_t UNDER Employee
   INHERIT SELECT PRIVILEGES;
```

Definición del tipo de la tabla

La primera tabla con tipo creada en el ejemplo anterior es `BusinessUnit`. Esta tabla se define del (OF) tipo `BusinessUnit_t`, por lo que contendrá instancias de este tipo. Esto significa que tendrá una columna correspondiente a cada atributo del tipo estructurado `BusinessUnit_t` y una columna adicional denominada *columna de identificador de objetos*.

Denominación del identificador de objetos

Puesto que las tablas con tipo contienen objetos a los que otros objetos pueden hacer referencia, cada tabla con tipo tiene una columna de *identificador de objetos* como primera columna. En este ejemplo, el tipo de la columna de identificador de objetos es (`BusinessUnit_t`). Puede asignar un nombre a la columna de identificador de objetos mediante la cláusula REF IS ... USER GENERATED. En este caso, la columna se denomina `Oid`. La parte USER GENERATED de la cláusula REF IS indica que se debe proporcionar el valor inicial de la columna de identificador de objetos de cada fila nueva que se inserte. En el diseño orientado al objeto, es práctica común separar por completo los datos del identificador de objeto. Por este motivo, no se puede actualizar el valor del identificador de objeto

una vez insertado dicho identificador. Si desea que DB2 genere los valores OID, puede utilizar una secuencia (SEQUENCE) o la función GENERATE_UNIQUE().

Especificación de la posición en la jerarquía de tablas

La tabla con tipo `Person` es del tipo `Person_t`. Para almacenar instancias de los subtipos de empleados y estudiantes, es necesario crear subtablas de la tabla `Person`, `Employee` y `Student`. Los dos subtipos adicionales de `Employee_t` también requieren tablas. Estas subtablas se denominan `Manager` y `Architect`. Al igual que un subtipo hereda los atributos de su supertipo, una subtabla hereda las columnas de su supertabla, incluida la columna de identificador de objetos.

Nota: Una subtabla debe residir en el mismo esquema que su supertabla.

Por consiguiente, las filas de la subtabla `Employee` tendrán un total de siete columnas: `Oid`, `Name`, `Age`, `Address`, `SerialNum`, `Salary` y `Dept`.

Una sentencia SELECT, UPDATE o DELETE que actúa sobre una supertabla, por omisión también actúa automáticamente sobre todas las subtablas de la misma. Por ejemplo, una sentencia UPDATE sobre la tabla `Employee` puede afectar a filas de las tablas `Employee`, `Manager` y `Architect`, pero una sentencia UPDATE sobre la tabla `Manager` sólo puede afectar a las filas de `Manager`.

Si desea restringir las acciones de la sentencia SELECT, INSERT o DELETE de forma que sólo se apliquen a la tabla especificada, utilice la opción ONLY.

Indicación de que los privilegios de SELECT se heredan

La cláusula obligatoria INHERIT SELECT PRIVILEGES de la sentencia CREATE TABLE especifica que a la subtabla resultante, como por ejemplo `Employee`, pueden acceder inicialmente los mismos usuarios y grupos que a la supertabla, como por ejemplo `Person`, a partir de la que se ha creado utilizando la cláusula UNDER. A cualquier usuario o grupo que actualmente tenga privilegios de SELECT sobre la supertabla se le otorgan privilegios de SELECT sobre la subtabla recién creada. El creador de la subtabla es el otorgante de los privilegios de SELECT. Para especificar privilegios tales como DELETE y UPDATE sobre las subtablas, debe emitir las mismas sentencias explícitas GRANT o REVOKE que utiliza para especificar privilegios sobre las tablas normales.

Los privilegios se pueden otorgar y revocar de forma independiente en cada nivel de una jerarquía de tablas. Si crea una subtabla, también puede revocar los privilegios de SELECT heredados sobre dicha subtabla. El hecho de revocar los privilegios de SELECT heredados de la subtabla impide que los usuarios que tienen privilegios de SELECT sobre la supertabla vean las columnas que sólo aparezcan en la subtabla. Revocando los privilegios de SELECT heredados por la subtabla, limita que sólo los usuarios que tengan privilegios de SELECT sobre la supertabla vean las columnas de la supertabla de las filas de la subtabla. Los usuarios sólo pueden actuar directamente sobre una subtabla si tienen el privilegio necesario sobre dicha subtabla. Por ello, para impedir que los usuarios seleccionen las bonificaciones de los directores en la subtabla, revoque el privilegio SELECT sobre dicha tabla y otórguelo únicamente a aquellos usuarios que necesiten esta información.

Definición de opciones de columnas

La cláusula WITH OPTIONS le permite definir opciones que se apliquen a una columna individual de la tabla con tipo. El formato de WITH OPTIONS es:

```
nombre-columna WITH OPTIONS opciones-columna
```

donde *nombre-columna* representa el nombre de la columna en la sentencia CREATE TABLE o ALTER TABLE, y *opciones-columna* representa las opciones definidas para la columna.

Por ejemplo, para impedir que los usuarios inserten valores nulos en una columna `SerialNum`, especifique la opción de columna NOT NULL del modo siguiente:

```
(SerialNum WITH OPTIONS NOT NULL)
```

Definición del ámbito de una columna de referencia

Otro uso de WITH OPTIONS es para especificar el ámbito (SCOPE) de una columna. Por ejemplo, en la tabla `Employee` y sus subtablas, la cláusula:

```
Dept WITH OPTIONS SCOPE BusinessUnit
```

declara que la columna `Dept` de esta tabla y sus subtablas tienen un *ámbito* de `BusinessUnit`. Esto significa que los valores de referencia de esta columna de la tabla `Employee` están destinados a hacer referencia a objetos de la tabla `BusinessUnit`.

Por ejemplo, la consulta siguiente sobre la tabla `Employee` utiliza el operador para deshacer referencias con el fin de indicar a DB2 que siga la vía de acceso que va de la columna `Dept` a la tabla `BusinessUnit`. El operador para deshacer referencias devuelve el valor de la columna `Nombre`:

```
SELECT Name, Salary, Dept->Name
   FROM Employee;
```

Conceptos relacionados:

- “Tipos estructurados definidos por el usuario” en la página 266
- “Jerarquías de tipos estructurados” en la página 268
- “Tablas con tipo” en la página 276

Tareas relacionadas:

- “Creación de tipos estructurados” en la página 266
- “Almacenamiento de objetos en filas de tablas con tipo” en la página 281
- “Eliminación de tablas con tipo” en la página 279
- “Definición de identificadores de objetos generados por el sistema” en la página 283
- “Definición de restricciones sobre columnas de identificador de objetos” en la página 285

Información relacionada:

- “Sentencia CREATE TABLE” en la publicación *Consulta de SQL, Volumen 2*
- “Sentencia CREATE TYPE (Estructurado)” en la publicación *Consulta de SQL, Volumen 2*

Eliminación de tablas con tipo

La eliminación de una tabla con tipo es parecida a la de una tabla sin tipo. Una diferencia importante es que el usuario se tiene que asegurar que la tabla que está

eliminando no tiene ninguna subtabla. Si tiene alguna, se producirá un error. El ejemplo siguiente muestra cómo eliminar la tabla Architect:

```
DROP TABLE Architect;
```

Cuando se elimina una subtabla de una jerarquía de tablas, las columnas asociadas a la subtabla dejan de estar accesibles. Mediante la posibilidad de sustitución, la eliminación de una subtabla tiene el efecto semántico de suprimir de las supertablas todas las filas de la subtabla. Esto puede dar como resultado la activación de activadores o restricciones de la integridad referencial que se hayan definido en las supertablas.

Otros objetos de base de datos, tales como tablas e índices, no se verán afectados aunque los paquetes y las sentencias dinámicas en antememoria se marquen como no válidos.

También se puede eliminar una jerarquía de tablas entera. Basta con añadir la cláusula HIERARCHY a la sentencia DROP TABLE y denominar la tabla raíz de la jerarquía. Por ejemplo:

```
DROP TABLE HIERARCHY Person;
```

La eliminación de una jerarquía de tablas no dará como resultado la activación de activadores ni restricciones de la integridad referencial.

Conceptos relacionados:

- "Jerarquías de tipos estructurados" en la página 268
- "Tablas con tipo" en la página 276

Información relacionada:

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

Posibilidad de sustitución en las tablas con tipo

Cuando se aplica una sentencia SELECT, UPDATE o DELETE a una tabla con tipo, la operación se aplica a la tabla mencionada y a todas las subtablas de la misma. Por ejemplo, si crea una tabla con tipo a partir del tipo estructurado Person_t y selecciona todas las filas de dicha tabla, la aplicación puede recibir no sólo las instancias del tipo Person, sino información de Person sobre las instancias del subtipo Employee y de otros subtipos.

La propiedad de posibilidad de sustitución también se aplica a las subtablas creadas a partir de subtipos. Por ejemplo, las sentencias SELECT, UPDATE y DELETE para la subtabla Employee se aplican tanto al tipo Employee_t como a los subtipos de éste. De forma parecida, una columna definida con el tipo Address_t puede contener instancias de una dirección estadounidense o de una dirección brasileña. Sin embargo, esto no significa que la sentencia UPDATE pueda cambiar el tipo de una fila, por ejemplo, si se debe actualizar una fila Person_t con datos de Employee_t. Para que esto funcione, se tiene que haber suprimido la fila Person_t e insertado la fila Employee_t como un nuevo tipo.

Para restringir la posibilidad de sustitución en las sentencias SELECT, UPDATE o DELETE, puede utilizar la cláusula ONLY. Por ejemplo, UPDATE ONLY(Person) SET sólo actualizará filas de la tabla Person y no de sus subtablas.

En cambio, las operaciones INSERT sólo se aplican a la tabla que se especifica en la propia sentencia INSERT. El hecho de realizar una inserción en la tabla `Employee` crea un objeto `Employee_t` en la jerarquía de tipos `Person`.

También puede sustituir instancias de subtipo cuando pase tipos estructurados a funciones en forma de parámetros, o como resultado de una función. Si una función tiene un parámetro del tipo `Address_t`, puede pasar una instancia de uno de sus subtipos, como por ejemplo, `US_addr_t`, en lugar de una instancia de `Address_t`. Las funciones de tabla externas no pueden devolver columnas de tipo estructurado.

Puesto que una columna o tabla se define con un tipo pero puede contener instancias de subtipos, a veces es importante distinguir entre el tipo que se ha utilizado para la definición y el tipo de la instancia que se devuelve realmente durante la ejecución. La definición del tipo estructurado en una columna, en una fila o en un parámetro de función se denomina *tipo estático*. El tipo real de una instancia de tipo estructurado se denomina *tipo dinámico*. Para recuperar información sobre el tipo dinámico, la aplicación puede utilizar las funciones incorporadas TYPE_NAME, TYPE_SCHEMA y TYPE_ID.

Conceptos relacionados:

- "Jerarquías de tipos estructurados" en la página 268
- "Tablas con tipo" en la página 276

Tareas relacionadas:

- "Creación de una jerarquía de tipos estructurados" en la página 270
- "Emisión de consultas para deshacer referencias" en la página 294

Almacenamiento de objetos en filas de tablas con tipo

Si los objetos se almacenan como filas de una tabla, cada columna de la tabla contiene un atributo del objeto. Al igual que en las tablas sin tipo, debe proporcionar datos para todas las columnas que estén definidas como NOT NULL, incluida la columna de identificador de objeto. Puesto que la columna de identificador de objetos es de tipo REF, que es un tipo muy firme, debe convertir el tipo de datos de los valores de identificador de objetos proporcionados por el usuario, utilizando para ello la función de conversión del tipo de datos generada por el sistema al crear el tipo estructurado (que se creó cuando el usuario creó el tipo estructurado). Por ejemplo, se puede almacenar una instancia de una persona en una tabla que contiene una columna para el nombre y una columna para la edad. En primer lugar mostramos un ejemplo de una sentencia CREATE TABLE para almacenar instancias de `Person`.

```
CREATE TABLE Person OF Person_t
   (REF IS Oid USER GENERATED)
```

Para insertar una instancia de `Person` en la tabla, puede utilizar la sintaxis siguiente:

```
INSERT INTO Person (Oid, Name, Age)
   VALUES(Person_t('a'), 'Andrew', 29);
```

Tabla 31. Tabla con tipo Person

Oid	Name	Age	Address
a	Andrew	29	

El programa accede a los atributos del objeto accediendo a las columnas de la tabla con tipo:

```
UPDATE Person
   SET Age=30
   WHERE Name='Andrew';
```

Después de la sentencia UPDATE anterior, el aspecto de la tabla es el siguiente:

Tabla 32. Tabla con tipo Person después de la actualización

Oid	Name	Age	Address
a	Andrew	30	

Puesto que existe un subtipo de `Person_t` denominado `Employee_t`, las instancias de `Employee_t` no se pueden almacenar en la tabla `Person`, y es necesario almacenarlas en una subtabla. La sentencia CREATE TABLE siguiente crea la subtabla `Employee` bajo la tabla `Person`:

```
CREATE TABLE Employee OF Employee_t UNDER Person
   INHERIT SELECT PRIVILEGES
   (SerialNum WITH OPTIONS NOT NULL,
   Dept WITH OPTIONS SCOPE BusinessUnit);
```

Y, de nuevo, una inserción en la tabla `Employee` tiene el aspecto siguiente:

```
INSERT INTO Employee (Oid, Name, Age, SerialNum, Salary)
   VALUES (Employee_t('s'), 'Susan', 39, 24001, 37000.48)
```

Tabla 33. Subtabla con tipo Employer

Oid	Name	Age	Address	SerialNum	Salary	Dept
s	Susan	39		24001	37000.48	

Si ejecuta la consulta siguiente, se le devolverá la información sobre Susan:

```
SELECT *
   FROM Employee
   WHERE Name='Susan';
```

Puede acceder a instancias de empleados y de personas, simplemente, mediante la ejecución de la sentencia de SQL sobre la tabla `Person`. Esta característica se denomina *posibilidad de sustitución*. Ejecutando una consulta sobre la tabla que contiene instancias que están más arriba en la jerarquía de tipos, automáticamente se obtienen instancias de tipos que están más abajo en la jerarquía. En otras palabras, la tabla `Person` tiene lógicamente el aspecto siguiente para las sentencias SELECT, UPDATE y DELETE:

Tabla 34. La tabla Person contiene instancias de Person y Employee

Oid	Name	Age	Address
a	Andrew	30	(nula)
s	Susan	39	(nula)

Si ejecuta la consulta siguiente, obtendrá un identificador de objeto e información de `Person_t` relativa tanto a Andrew (una persona) como a Susan (una empleada):

```
SELECT *
   FROM Person;
```

Conceptos relacionados:

- "Relaciones entre objetos de las tablas con tipo" en la página 287
- "Posibilidad de sustitución en las tablas con tipo" en la página 280
- "Tablas con tipo" en la página 276

Tareas relacionadas:
- "Almacenamiento de instancias de tipos estructurados" en la página 267
- "Creación de tablas con tipo" en la página 276

Definición de identificadores de objetos generados por el sistema

Existen dos enfoques frecuentes para generar valores exclusivos, pudiéndose aplicar ambos a los identificadores de objetos:
- con secuencias
- con la función GENERATE_UNIQUE

Si tiene necesidad de utilizar valores numéricos como identificadores de objetos, puede utilizar una secuencia. Para empezar, utilice la cláusula REF USING para especificar que el tipo base de la referencia al objeto va a ser de tipo numérico, en el caso siguiente un entero (INT):

```
CREATE TYPE BusinessUnit_t AS
  (Name VARCHAR(20),
   Headcount INT)
   REF USING INT
   MODE DB2SQL
```

La definición de la tabla con tipo es la siguiente:

```
CREATE TABLE BusinessUnit OF BusinessUnit_t
  (REF IS oid USER GENERATED)
```

La secuencia para generar identificadores de objetos se puede definir del modo siguiente:

```
CREATE SEQUENCE BusinessUnitOid AS REF(BusinessUnit_t)
```

Observe que la modificación de los datos de una subtabla modifica implícitamente todas las supertablas. Por consiguiente, es mejor añadir el activador que invoca la secuencia para generar el identificador de objetos a la raíz de la jerarquía de tablas.

```
CREATE TRIGGER Gen_Bunit_oid
  NO CASCADE
  BEFORE INSERT ON BusinessUnit
  REFERENCING NEW AS new
  FOR EACH ROW
  MODE DB2SQL
  SET new.oid = NEXTVAL FOR BusinessUnitOid
```

Observe que, puesto que la secuencia se ha definido como REF(BusinessUnitOid), no se requiere ninguna conversión del tipo de datos para asignarlo a la columna Oid.

Ahora se puede añadir una nueva unidad de negocio:

```
INSERT INTO BusinessUnit (Name, Headcount)
  VALUES('Software', 10)
```

El uso de una secuencia también permite recuperar el identificador de objetos generado y utilizarlos en sentencias posteriores. Por ejemplo, puede añadir un empleado a Software BusinessUnit suponiendo que la columna Dept es del tipo REF(BusinessUnit):

```
INSERT INTO Employee(Name, Age, SerialNum, Salary, Dept)
  VALUES('Tom', 28, 106, 60000, PREVVAL FOR BusinessUnitOid)
```

Como alternativa al uso de secuencias para generar identificadores de objetos, puede utilizar la función GENERATE_UNIQUE. Puesto que GENERATE_UNIQUE devuelve un valor CHAR (13) FOR BIT DATA, asegúrese de que la cláusula REF USING de la sentencia CREATE TYPE puede acomodar un valor de este tipo. El valor por omisión, VARCHAR (16) FOR BIT DATA, resulta adecuado para estos propósitos. Por ejemplo, suponga que el tipo `BusinessUnit_t` se crea con el tipo de representación por omisión; es decir, que no se especifica la cláusula REF USING, del modo siguiente:

```
CREATE TYPE BusinessUnit_t AS
   (Name VARCHAR(20),
   Headcount INT)
   MODE DB2SQL;
```

La definición de la tabla con tipo es la siguiente:

```
CREATE TABLE BusinessUnit OF BusinessUnit_t
(REF IS Oid USER GENERATED);
```

Observe que siempre se proporciona la cláusula USER GENERATED.

Así, una sentencia INSERT para insertar una fila en una tabla con tipo puede tener el aspecto siguiente:

```
INSERT INTO BusinessUnit (Oid, Name, Headcount)
   VALUES(BusinessUnit_t(GENERATE_UNIQUE( )), 'Toy' 15);
```

Para insertar un empleado que pertenezca al departamento `Toy`, puede utilizar una sentencia como la siguiente, que emite una subselección para recuperar el valor de la columna de identificador de objetos de la tabla `BusinessUnit`, convierte el tipo de datos al tipo `BusinessUnit_t` e inserta este valor en la columna `Dept`:

```
INSERT INTO Employee (Oid, Name, Age, SerialNum, Salary, Dept)
   VALUES(Employee_t('d'), 'Dennis', 26, 105, 30000,
     BusinessUnit_t(SELECT Oid FROM BusinessUnit WHERE Name='Toy'));
```

En lugar de insertar explícitamente el identificador de objetos generado en la sentencia INSERT, puede encapsular la generación e inserción del identificador de objetos en un activador. Un activador de la raíz de la jerarquía puede automatizar la invocación de la función GENERATE_UNIQUE. El activador siguiente generará identificadores para inserciones en las tablas Person, Employee, Architect y Manager.

```
CREATE TRIGGER Gen_Person_oid
  NO CASCADE
  BEFORE INSERT ON Person
  REFERENCING NEW AS new
  FOR EACH ROW
  MODE DB2SQL
  SET new.oid = Person_t (generate_unique());
```

Conceptos relacionados:

- "Tipos de referencia" en la página 286
- "Relaciones entre objetos de las tablas con tipo" en la página 287

Tareas relacionadas:

- "Creación de una jerarquía de tipos estructurados" en la página 270
- "Emisión de consultas para deshacer referencias" en la página 294
- "Creación de tablas con tipo" en la página 276

Información relacionada:

- "Sentencia CREATE TRIGGER" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE TYPE (Estructurado)" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE SEQUENCE" en la publicación *Consulta de SQL, Volumen 2*

Definición de restricciones sobre columnas de identificador de objetos

Si desea utilizar la columna de identificador de objetos como columna de clave de la tabla padre en una clave foránea, antes debe modificar la tabla con tipo para añadir una restricción explícita de clave primaria o exclusiva sobre la columna de identificador de objetos. Por ejemplo, suponga que desea crear una relación autorreferente sobre los empleados en que el director de cada empleado tiene que existir siempre como empleado en la tabla de empleados, tal como se muestra en la Figura 7.

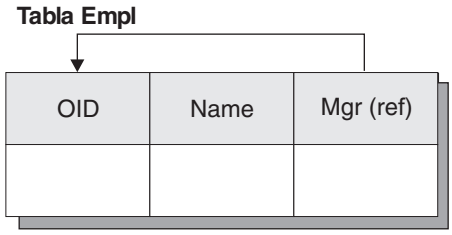


Figura 7. Ejemplo de tipo autorreferente

Para definir restricciones sobre una columna de identificador de objetos con el fin de crear una relación de autorreferencia sobre un objeto:

Paso 1. Cree el tipo, por ejemplo:

```
CREATE TYPE Empl_t AS
   (Name VARCHAR(10), Mgr REF(Empl_t))
   MODE DB2SQL;
```

Paso 2. Cree la tabla con tipo, por ejemplo:

```
CREATE TABLE Empl OF Empl_t
   (REF IS Oid USER GENERATED);
```

Paso 3. Añada la restricción primaria o exclusiva sobre la columna `Oid`, por ejemplo:

```
ALTER TABLE Empl ADD CONSTRAINT pk1 UNIQUE(Oid);
```

Paso 4. Añada la restricción de clave foránea, por ejemplo:

```
ALTER TABLE Empl ADD CONSTRAINT fk1 FOREIGN KEY(Mgr)
   REFERENCES Empl (Oid);
```

Conceptos relacionados:

- "Tipos de referencia" en la página 286

Tareas relacionadas:

- "Creación de tipos estructurados" en la página 266
- "Almacenamiento de objetos en filas de tablas con tipo" en la página 281
- "Definición de identificadores de objetos generados por el sistema" en la página 283

Información relacionada:
- "Sentencia CREATE TABLE" en la publicación *Consulta de SQL, Volumen 2*
- "Sentencia CREATE TYPE (Estructurado)" en la publicación *Consulta de SQL, Volumen 2*

Tipos de referencia

Tipos de referencia

Para cada tipo estructurado que crea el usuario, DB2® crea automáticamente un tipo compañero. El tipo compañero se denomina *tipo de referencia* y el tipo estructurado al que hace referencia se denomina *tipo referenciado*. Las tablas con tipo pueden hacer un uso especial del tipo de referencia. También puede utilizar tipos de referencia en las sentencias de SQL, del mismo modo que utiliza otros tipos definidos por el usuario. Para utilizar un tipo de referencia en una sentencia de SQL, use REF(nombre-tipo), donde nombre-tipo representa al tipo referenciado.

DB2 utiliza el tipo de referencia como tipo de la columna de identificador de objetos de las tablas con tipo. El identificador de objetos identifica de forma exclusiva un objeto de fila en la jerarquía de la tabla con tipo. Asimismo, DB2 utiliza tipos de referencia para almacenar referencias a filas de las tablas con tipo. Puede utilizar tipos de referencia para hacer referencia a cada uno de los objetos de fila de la tabla.

Las referencias tienen tipos firmes. Por consiguiente, se tiene que disponer de una manera de utilizar el tipo en las expresiones. Cuando cree el tipo raíz de una jerarquía de tipos, puede especificar el tipo base para una referencia con la cláusula REF USING de la sentencia CREATE TYPE. El tipo base para una referencia se denomina *tipo de representación*. Si no se especifica el tipo de representación mediante la cláusula REF USING, DB2 utiliza el tipo de datos por omisión VARCHAR(16) FOR BIT DATA. Todos los subtipos del tipo raíz heredan el tipo de representación del mismo. La cláusula REF USING sólo es válida cuando se define el tipo raíz de una jerarquía. En los ejemplos que se utilizan en este apartado, el tipo de representación para el tipo `BusinessUnit_t` es INTEGER, mientras que el tipo de representación para `Person_t` es VARCHAR(13).

Conceptos relacionados:
- "Integridad referencial frente a referencias con ámbito" en la página 290
- "Relaciones entre objetos de las tablas con tipo" en la página 287
- "Tablas con tipo" en la página 276

Tareas relacionadas:
- "Almacenamiento de objetos en filas de tablas con tipo" en la página 281
- "Emisión de consultas para deshacer referencias" en la página 294
- "Definición de identificadores de objetos generados por el sistema" en la página 283
- "Definición de restricciones sobre columnas de identificador de objetos" en la página 285

- "Creación de tablas con tipo" en la página 276

Relaciones entre objetos de las tablas con tipo

Puede definir relaciones entre objetos de una tabla con tipo y objetos de otra tabla. También puede definir relaciones entre objetos de la misma tabla con tipo. Por ejemplo, suponga que ha definido una tabla con tipo que contiene instancias de departamentos. En lugar de mantener los números de departamento en la tabla `Employee`, la columna `Dept` de la tabla `Employee` puede contener un puntero lógico a uno de los departamentos de la tabla `BusinessUnit`. Estos punteros se denominan *referencias* y se ilustran en la Figura 8.

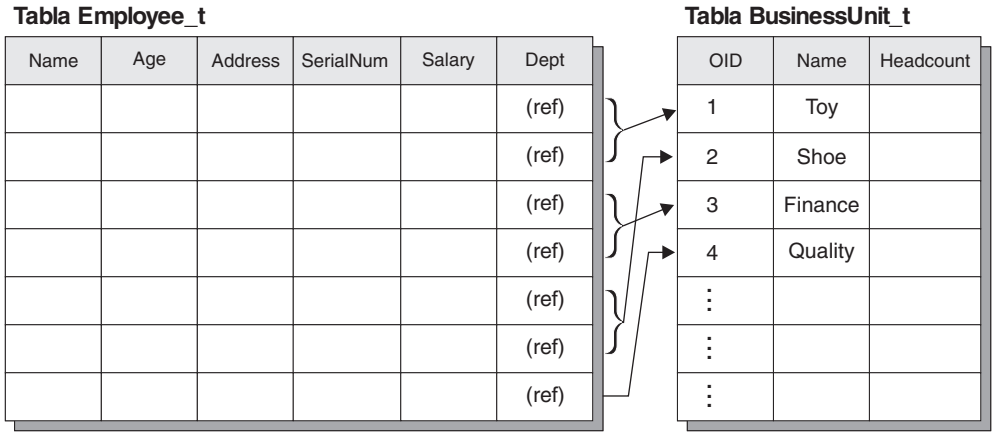


Figura 8. Referencias de tipo estructurado de Employee_t a BusinessUnit_t

Una tabla normal (que no es una tabla con tipo) puede tener una columna REF que haga referencia a una tabla con tipo. Sin embargo, una tabla con tipo no puede tener una columna REF que apunte a una tabla normal.

Importante: Las referencias no realizan la misma función que las restricciones referenciales. Es posible tener una referencia a un departamento que no exista. Si es importante mantener la integridad entre departamento y empleados, puede definir una restricción referencial entre estas dos tablas. La potencia real de las referencias consiste en que brindan la posibilidad de escribir consultas que naveguen por la relación entre las tablas. Lo que hace la consulta es deshacer la referencia de relación y crear una instancia del objeto a la que se apunte. El operador que se utiliza para realizar esta acción recibe el nombre de *operador para deshacer referencias* y tiene el aspecto siguiente: ->.

Por ejemplo, la consulta siguiente sobre la tabla `Employee` utiliza el operador para deshacer referencias con el fin de indicar a DB2® que siga la vía de acceso que va de la columna `Dept` a la tabla `BusinessUnit`. El operador para deshacer referencias devuelve el valor de la columna `Nombre`:

```
SELECT Name, Salary, Dept->Name
   FROM Employee;
```

Conceptos relacionados:

- "Tipos de referencia" en la página 286
- "Integridad referencial frente a referencias con ámbito" en la página 290
- "Tablas con tipo" en la página 276

Tareas relacionadas:

- "Restricción de los tipos devueltos mediante un predicado TYPE" en la página 296
- "Definición de identificadores de objetos generados por el sistema" en la página 283

Definición de relaciones semánticas con referencias

Mediante la cláusula WITH OPTIONS de CREATE TABLE, se puede definir que existe una relación entre una columna de una tabla y los objetos de la misma tabla o de otra. La cláusula WITH OPTIONS de CREATE TABLE define las propiedades de columna para una columna de una tabla con tipo. Estas propiedades de tabla definibles incluyen la relación entre una columna de una tabla y los objetos de la misma tabla (o de otra). En el ejemplo que se ilustra a continuación, el departamento de cada empleado es en realidad una referencia a un objeto de la tabla `BusinessUnit`. Para definir los objetos de destino de una columna de referencia determinada, utilice la palabra clave SCOPE en la cláusula WITH OPTIONS.

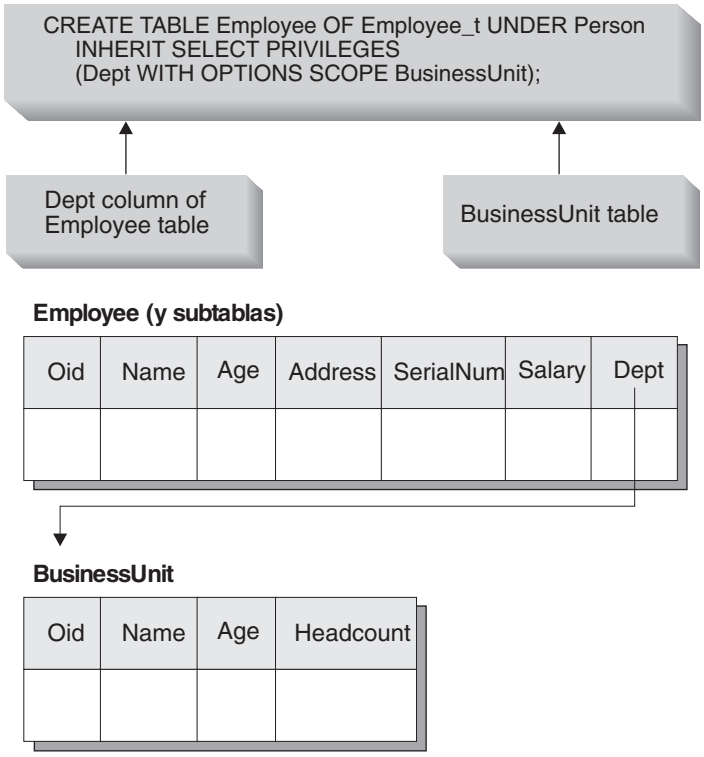


Figura 9. El atributo Dept hace referencia a un objeto de BusinessUnit

Relaciones de autorreferencia

También puede definir referencias con ámbito a objetos de la misma tabla con tipo. Las sentencias contenidas en el ejemplo siguiente crean una tabla con tipo para piezas y una tabla con tipo para proveedores. Para mostrar las definiciones de tipos de referencia, el ejemplo incluye también las sentencias utilizadas para crear los tipos:

```
CREATE TYPE Company_t AS
   (name VARCHAR(30),
   location VARCHAR(30))
   MODE DB2SQL
```

```
CREATE TYPE Part_t AS
   (Descript VARCHAR(20),
   Supplied_by REF(Company_t),
   Used_in REF(part_t))
   MODE DB2SQL

CREATE TABLE Suppliers OF Company_t
   (REF IS suppno USER GENERATED)

CREATE TABLE Parts OF Part_t
   (REF IS Partno USER GENERATED,
   Supplied_by WITH OPTIONS SCOPE Suppliers,
   Used_in WITH OPTIONS SCOPE Parts)
```

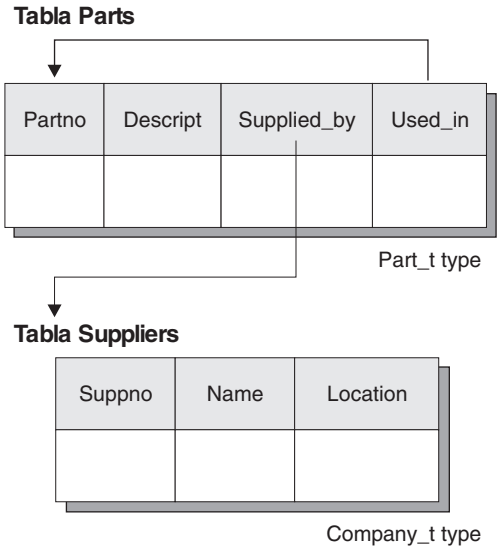


Figura 10. Ejemplo de un ámbito autorreferente

Puede utilizar referencias con ámbito para escribir consultas que, sin las referencias con ámbito, se tendrían que escribir como uniones externas o subconsultas correlacionadas. Por ejemplo, las dos consultas siguientes recuperan el proveedor de la pieza en que se utiliza la pieza '1234':

```
SELECT Used_in->Supplied_by->Name
  FROM Parts
  WHERE Partno = Part_t('1234')
```

Sin una referencia con ámbito, la consulta tiene el aspecto siguiente:

```
SELECT S.Name
  FROM (Parts AS P RIGHT OUTER JOIN Parts C ON P.Used_in = C.Partno)
    RIGHT OUTER JOIN Suppliers S ON C.Supplied_by = S.Suppno
  WHERE P.Partno = Part_t('1234')
```

Conceptos relacionados:

- "Tipos de referencia" en la página 286
- "Integridad referencial frente a referencias con ámbito" en la página 290
- "Relaciones entre objetos de las tablas con tipo" en la página 287
- "Tablas con tipo" en la página 276

Tareas relacionadas:

- "Definición de identificadores de objetos generados por el sistema" en la página 283

Integridad referencial frente a referencias con ámbito

Aunque las referencias con ámbito definen relaciones entre objetos de las tablas, son distintas de las relaciones de integridad referencial. Los ámbitos proporcionan información, simplemente, sobre una tabla de destino. Esta información se utiliza cuando se deshacen referencias de objetos de la tabla de destino. Las referencias con ámbito no requieren ni imponen que exista un valor en la otra tabla. Para asegurarse de que los objetos de estas relaciones existen, debe añadir una restricción referencial.

Conceptos relacionados:

- "Tipos de referencia" en la página 286
- "Tablas con tipo" en la página 276

Tareas relacionadas:

- "Definición de relaciones semánticas con referencias" en la página 288

Vistas con tipo

Vistas con tipo

Para las vistas con tipo, los nombres y tipos de datos de los atributos del tipo estructurado se convierten en nombres y tipos de datos de las columnas de esta vista con tipo. Las filas de la vista con tipo se pueden contemplar como una representación de instancias del tipo estructurado.

Al igual que una tabla con tipo, una vista con tipo puede formar parte de una jerarquía de vistas. Una subvista hereda columnas de su supervista. El término subvista se aplica a todas las vistas con tipo que están por debajo de una vista con tipo en la jerarquía de vistas. Una subvista adecuada de una vista V es una vista que está por debajo de V en la jerarquía de vistas con tipo.

Conceptos relacionados:

- "Tipos estructurados definidos por el usuario" en la página 266
- "Tablas con tipo" en la página 276

Tareas relacionadas:

- "Creación de vistas con tipo" en la página 291
- "Modificación de vistas con tipo" en la página 293
- "Eliminación de vistas con tipo" en la página 293

Información relacionada:

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

Creación de vistas con tipo

Puede crear una vista con tipo mediante la sentencia CREATE VIEW. Por ejemplo, para crear una vista de la tabla con tipo `BusinessUnit`, puede definir un tipo estructurado que tenga los atributos deseados y, a continuación, crear una vista con tipo utilizando dicho tipo:

```
CREATE TYPE VBusinessUnit_t AS (Name VARCHAR(20))
   MODE DB2SQL;

CREATE VIEW VBusinessUnit OF VBusinessUnit_t MODE DB2SQL
   (REF IS VObjectID USER GENERATED)
   AS SELECT VBusinessUnit_t(VARCHAR(Oid)), Name FROM BusinessUnit;
```

La cláusula OF de la sentencia CREATE VIEW indica a DB2 que base las columnas de la vista en los atributos del tipo estructurado indicado. En este caso, DB2 basa las columnas de la vista en el tipo estructurado `VBusinessUnit_t`.

La columna `VObjectID` de la vista tiene el tipo REF(VBusinessUnit_t). Puesto que no se puede realizar una conversión del tipo de datos REF(BusinessUnit_t) al tipo de datos REF(VBusinessUnit_t), antes debe convertir el tipo de datos del valor de la columna `Oid` de la tabla `BusinessUnit` al tipo de datos VARCHAR, y luego convertir el tipo de datos VARCHAR al tipo de datos REF(VBusinessUnit_t).

La cláusula MODE DB2SQL especifica la modalidad de la vista con tipo. Ésta es la única modalidad que se soporta actualmente.

La cláusula REF IS... es idéntica a la de la sentencia CREATE TABLE con tipo. Proporciona un nombre para la columna de identificador de objetos de la vista (en este caso, `VObjectID`), que es la primera columna de la vista. Si crea una vista raíz, debe especificar una columna de identificador de objetos para la vista. Si crea una subvista, ésta hereda la columna de identificador de objetos.

La cláusula USER GENERATED especifica que el usuario debe proporcionar el valor para la columna de identificador de objetos cuando inserte una fila. Una vez insertada, la columna de identificador de objetos no se puede actualizar.

El cuerpo de la vista, que sigue a la palabra clave AS, es una sentencia SELECT que determina el contenido de la vista. Los tipos de columna devueltos por esta sentencia SELECT tienen que ser compatibles con los tipos de columna de la vista con tipo, incluida la columna de identificador de objetos.

Para ilustrar la creación de una jerarquía de vistas con tipo, el ejemplo siguiente define una jerarquía de vistas que omite algunos datos delicados y suprime algunas distinciones de tipo de la jerarquía de tablas `Person`:

```
CREATE TYPE VPerson_t AS (Name VARCHAR(20))
   MODE DB2SQL;

CREATE TYPE VEmployee_t UNDER VPerson_t
   AS (Salary INT, Dept REF(VBusinessUnit_t))
   MODE DB2SQL;

CREATE VIEW VPerson OF VPerson_t MODE DB2SQL
   (REF IS VObjectID USER GENERATED)
   AS SELECT VPerson_t (VARCHAR(Oid)), Name FROM ONLY(Person);

CREATE VIEW VEmployee OF VEmployee_t MODE DB2SQL
   UNDER VPerson INHERIT SELECT PRIVILEGES
```

```
(Dept WITH OPTIONS SCOPE VBusinessUnit)
AS SELECT VEmployee_t(VARCHAR(Oid)), Name, Salary,
   VBusinessUnit_t(VARCHAR(Dept))
FROM Employee;
```

Las dos sentencias CREATE TYPE crean los tipos estructurados que se necesitan para crear la jerarquía de vistas de objetos para este ejemplo.

La primera sentencia CREATE VIEW con tipo anterior crea la vista raíz de la jerarquía, `VPerson`, y es muy parecida a la definición de la vista `VBusinessUnit`. La diferencia reside en el uso de `ONLY(Person)` para asegurarse de que sólo las filas de la jerarquía de tablas `Person` que aparezcan en la tabla `Person`, y no en ninguna subtabla, se incluyan en la vista `VPerson`. Así se asegurará de que los valores de `Oid` en `VPerson` sean exclusivos en comparación con los valores de `Oid` en `VEmployee`. La segunda sentencia CREATE VIEW crea una subvista `VEmployee` bajo la vista `VPerson`. Tal como en el caso de la cláusula UNDER en la sentencia CREATE TABLE...UNDER, la cláusula UNDER establece la jerarquía de vistas. Debe crear una subvista en el mismo esquema que su supervista. Al igual que las tablas con tipo, las subvistas heredan columnas de su supervista. Las filas de la vista `VEmployee` heredan las columnas `VObjectID` y `Name` de `VPerson` y tienen las columnas adicionales `Salary` y `Dept` asociadas al tipo `VEmployee_t`.

La cláusula INHERIT SELECT PRIVILEGES tiene el mismo efecto cuando se emite una sentencia CREATE VIEW que cuando se emite una sentencia CREATE TABLE con tipo. La cláusula WITH OPTIONS en la definición de una vista con tipo también tiene el mismo efecto que en la definición de una tabla con tipo. La cláusula WITH OPTIONS permite especificar opciones de columnas, como por ejemplo SCOPE. La cláusula READ ONLY impone que una columna de supervista se marque como sólo de lectura, de forma que las definiciones de subvistas posteriores puedan especificar una expresión para la misma columna que también sea sólo de lectura.

Si una vista tiene una columna de referencia, tal como la columna `Dept` de la vista `VEmployee`, debe asociar un ámbito a la columna para utilizarla en operaciones para deshacer referencias de SQL. Si no se especifica un ámbito para la columna de referencia de la vista y la columna de tabla o vista subyacente tiene un ámbito, el ámbito de la columna subyacente se pasa a la columna de referencia de la vista. Puede asignar explícitamente un ámbito a la columna de referencia de la vista mediante la cláusula WITH OPTIONS. En el ejemplo anterior, la columna `Dept` de la vista `VEmployee` recibe la vista `VBusinessUnit` como ámbito. Si la columna de tabla o vista subyacente no tiene un ámbito, y si no se ha asignado explícitamente un ámbito en la definición de la vista, o si no se ha asignado un ámbito mediante una sentencia ALTER VIEW, la columna de referencia permanece sin ámbito.

Conceptos relacionados:

- "Tipos estructurados definidos por el usuario" en la página 266
- "Tablas con tipo" en la página 276
- "Vistas con tipo" en la página 290

Información relacionada:

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

Modificación de vistas con tipo

La sentencia ALTER VIEW modifica una vista existente alterando una columna de tipo de referencia para añadir un ámbito. Cualquier otro cambio que pretenda realizar sobre una vista requiere que se elimine y se vuelva a crear la vista.

Cuando se modifica la vista, el ámbito se debe añadir a una columna de tipo de referencia existente que aún no tenga definido un ámbito. Además, la columna no se tiene que haber heredado de una supervista.

El tipo de datos del nombre de columna en la sentencia ALTER VIEW debe ser REF (tipo del nombre de tabla con tipo o del nombre de vista con tipo).

Conceptos relacionados:

- "Tipos estructurados definidos por el usuario" en la página 266
- "Tablas con tipo" en la página 276
- "Vistas con tipo" en la página 290

Tareas relacionadas:

- "Creación de vistas con tipo" en la página 291

Información relacionada:

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

Eliminación de vistas con tipo

El ejemplo siguiente muestra cómo eliminar EMP_VIEW:

```
DROP VIEW EMP_VIEW;
```

Las vistas que dependen de la vista eliminada pasan a estar inoperantes.

Otros objetos de base de datos, tales como tablas e índices, no se verán afectados aunque los paquetes y las sentencias dinámicas en antememoria se marquen como no válidos.

Tal como en el caso de una jerarquía de tablas, es posible eliminar una jerarquía de vistas entera en una sentencia mencionando la vista raíz de la jerarquía, como en el ejemplo siguiente:

```
DROP VIEW HIERARCHY VPerson;
```

Conceptos relacionados:

- "Tipos estructurados definidos por el usuario" en la página 266
- "Tablas con tipo" en la página 276
- "Vistas con tipo" en la página 290

Tareas relacionadas:

- "Creación de vistas con tipo" en la página 291

Información relacionada:

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

Consulta de tablas con tipo y vistas con tipo

Emisión de consultas para deshacer referencias

Siempre que tenga una referencia con ámbito, puede utilizar una *operación para deshacer referencias* para emitir consultas que, de otro modo, necesitarían uniones externas o subconsultas correlacionadas. Considere el atributo `Dept` de la tabla `Employee` y las subtablas de `Employee`, que tienen como ámbito la tabla `BusinessUnit`. El ejemplo siguiente devuelve los nombres, salarios y nombres de departamento, o valores nulos (NULL) si es pertinente, de todos los empleados de la base de datos; esto significa que la consulta devuelve estos valores para cada fila de la tabla `Employee` y de las subtablas de `Employee`. Puede escribir una consulta parecida utilizando una subconsulta correlacionada o una unión externa. Sin embargo, resulta más fácil utilizar el *operador para deshacer referencias* (->) para recorrer la vía de acceso desde la columna de referencia de la tabla `Employee` y de las subtablas a la tabla `BusinessUnit`, y devolver el resultado de la columna `Name` de la tabla `BusinessUnit`.

El formato simple de la operación para deshacer referencias es el siguiente:

```
expresión-referencia-con-ámbito->columna-de-tabla-con-tipo-de-destino
```

La consulta siguiente utiliza el operador para deshacer referencias para obtener la columna `Name` de la tabla `BusinessUnit`:

```
SELECT Name, Salary, Dept->Name
   FROM Employee
```

El resultado de la consulta es el siguiente:

```
NAME                 SALARY      NAME
-------------------- ----------- --------------------
Dennis               30000       Toy
Eva                  45000       Shoe
Franky               39000       Shoe
Iris                 55000       Toy
Christina            85000       Toy
Ken                  105000      Shoe
Leo                  92000       Shoe
Brian                112000      Toy
Susan                37000.48    ---
```

También puede deshacer referencias de referencias autorreferentes. Piense en la tabla Parts. La consulta siguiente lista las piezas utilizadas directamente en un ala con las ubicaciones de los proveedores de las piezas:

```
SELECT P.Descript, P.Supplied_by->Location
   FROM Parts P
   WHERE P.Used_in->Descript='Wing';
```

Función incorporada DEREF

También puede deshacer referencias para obtener objetos estructurados completos con un solo valor mediante la función incorporada DEREF. El formato simple de DEREF es el siguiente:

```
DEREF (expresión-referencia-con-ámbito)
```

DEREF se suele utilizar en el contexto de otras funciones incorporadas, como por ejemplo TYPE_NAME, o para obtener un objeto estructurado entero a efectos de enlazarlo lógicamente a una aplicación.

Otras funciones incorporadas relacionadas con el tipo

Muchas veces se invoca la función DEREF formando parte de las funciones incorporadas TYPE_NAME, TYPE_ID o TYPE_SCHEMA. El objetivo de estas funciones consiste, respectivamente, en devolver el nombre, el ID interno y el nombre de esquema del tipo dinámico de una expresión. Por ejemplo, el ejemplo siguiente crea una tabla con tipo `Project` con un atributo denominado `Responsible`:

```
CREATE TYPE Project_t
   AS (Projid INT, Responsible REF(Employee_t))
   MODE DB2SQL;

CREATE TABLE Project
   OF Project_t (REF IS Oid USER GENERATED,
   Responsible WITH OPTIONS SCOPE Employee);
```

El atributo `Responsible` se define como referencia a la tabla `Employee`, por lo que puede hacer referencia a instancias de directores y arquitectos, así como a empleados. Si la aplicación necesita conocer el nombre del tipo dinámico de cada fila, puede utilizar una consulta como la siguiente:

```
SELECT Projid, Responsible->Name,
   TYPE_NAME(DEREF(Responsible))
   FROM PROJECT;
```

En el ejemplo anterior se utiliza el operador para deshacer referencias a fin de devolver el valor de `Name` de la tabla `Employee`, y se invoca la función DEREF para devolver el tipo dinámico correspondiente a la instancia de `Employee_t`.

Requisito de autorización: Para utilizar la función DEREF, debe tener autorización de SELECT sobre cada una de las tablas y subtablas de la porción de la jerarquía de tablas a que se hace referencia. Por ejemplo, en la consulta anterior, se necesita el privilegio de SELECT sobre las tablas con tipo `Employee`, `Manager` y `Architect`.

Conceptos relacionados:

- "Tipos estructurados definidos por el usuario" en la página 266
- "Tipos de referencia" en la página 286
- "Relaciones entre objetos de las tablas con tipo" en la página 287
- "Tablas con tipo" en la página 276
- "Vistas con tipo" en la página 290

Tareas relacionadas:

- "Almacenamiento de objetos en filas de tablas con tipo" en la página 281
- "Devolución de objetos de un tipo determinado mediante ONLY" en la página 296
- "Restricción de los tipos devueltos mediante un predicado TYPE" en la página 296
- "Devolución de todos los tipos posibles mediante OUTER" en la página 297
- "Definición de identificadores de objetos generados por el sistema" en la página 283

Información relacionada:

- "Función escalar DEREF" en la publicación *Consulta de SQL, Volumen 1*

Devolución de objetos de un tipo determinado mediante ONLY

Para hacer que una consulta sólo devuelva objeto de un tipo determinado, y no los subtipos del mismo, utilice la palabra clave ONLY. Por ejemplo, la consulta siguiente sólo devuelve los nombres de los empleados que no son arquitectos ni directores:

```
SELECT Name
FROM ONLY(Employee);
```

La consulta anterior devuelve el resultado siguiente:

```
NAME
--------------------
Dennis
Eva
Franky
Susan
```

Para proteger la seguridad de los datos, el uso de ONLY requiere el privilegio de SELECT sobre cada una de las subtablas de `Employee`.

También puede utilizar la cláusula ONLY para restringir el funcionamiento de una sentencia UPDATE o DELETE a la tabla mencionada. Es decir, la cláusula ONLY asegura que no se realice la operación sobre ninguna subtabla de la tabla indicada.

Conceptos relacionados:

- "Tipos diferenciados definidos por el usuario" en la página 247
- "Tablas con tipo" en la página 276

Tareas relacionadas:

- "Almacenamiento de objetos en filas de tablas con tipo" en la página 281
- "Emisión de consultas para deshacer referencias" en la página 294

Restricción de los tipos devueltos mediante un predicado TYPE

Si desea disponer de una manera más general para restringir las filas que se devuelven o se ven afectadas por una sentencia de SQL, puede utilizar el predicado de tipo. El predicado de tipo permite comparar el tipo dinámico de una expresión con uno o más tipos indicados. Una versión simple del predicado de tipo es:

```
<expresión> IS OF (<nombre_tipo>[, ...])
```

donde *expresión* representa una expresión de SQL que devuelve una instancia de un tipo estructurado, y *nombre_tipo* representa uno o más tipos estructurados con los que se comparará la instancia.

Por ejemplo, la consulta siguiente devuelve el personal que tiene más de 35 años y que son directores o arquitectos:

```
SELECT Name
   FROM Employee E
   WHERE E.Age > 35 AND
   DEREF(E.Oid) IS OF (Manager_t, Architect_t);
```

La consulta anterior devuelve el resultado siguiente:

```
NAME
--------------------
Ken
```

Conceptos relacionados:

- "Tipos estructurados definidos por el usuario" en la página 266
- "Tipos de referencia" en la página 286
- "Tablas con tipo" en la página 276
- "Vistas con tipo" en la página 290

Tareas relacionadas:

- "Almacenamiento de objetos en filas de tablas con tipo" en la página 281
- "Emisión de consultas para deshacer referencias" en la página 294

Devolución de todos los tipos posibles mediante OUTER

Cuando DB2 devuelve el valor de una fila de tipo estructurado, la aplicación no sabe necesariamente qué atributos contiene o puede contener esa instancia en concreto. Por ejemplo, cuando se devuelve una persona, esa persona puede tener únicamente los atributos de una persona o puede tener los atributos de empleado, director u otro subtipo de persona. Si la aplicación necesita obtener los valores de todos los atributos posibles en una consulta de SQL, puede utilizar la palabra clave OUTER en la referencia a la tabla.

OUTER (*nombre-tabla*) y OUTER(*nombre-vista*) devuelven una tabla virtual que consta de las columnas de la tabla o vista, seguidas de las columnas adicionales introducidas por cada una de sus subtablas, si las hay. Las columnas adicionales se añaden a la parte derecha de la tabla, recorriendo la jerarquía de subtablas en orden de profundidad. Las subtablas que tienen un padre común se recorren en el orden en que se crearon sus tipos respectivos. Las filas incluyen todas las filas de *nombre-tabla* y todas las filas adicionales de las subtablas de *nombre-tabla*. Se devuelven valores nulos para las columnas que no se encuentran en la subtabla correspondiente a la fila.

Puede utilizar OUTER, por ejemplo, cuando desee ver información sobre el personal que tiende a superar la normalidad. La consulta siguiente devuelve información de la jerarquía de tablas `Person` que tiene un salario `Salary` elevado o un promedio de puntos `GPA` de alta nota:

```
SELECT *
   FROM OUTER(Person) P
   WHERE P.Salary > 200000
   OR P.GPA > 3.95 ;
```

El uso de OUTER(Person) permite hacer referencia a atributos de subtipo, cosa que no sería posible en las consultas de `Person`.

La utilización de OUTER requiere el privilegio de SELECT sobre cada una de las subtablas o vistas de la tabla referenciada, puesto que mediante este uso se expone toda la información contenida en las mismas.

Suponga que la aplicación necesita ver no sólo los atributos del personal que logra altos objetivos, sino cuál es el tipo más específico de cada uno de ellos. Lo puede hacer en una simple consulta pasando el identificador de objetos de un objeto a la función incorporada TYPE_NAME y combinándola con una consulta OUTER, del modo siguiente:

```
SELECT TYPE_NAME(DEREF(P.Oid)), P.*
   FROM OUTER(Person) P
   WHERE P.Salary > 200000 OR
   P.GPA > 3.95 ;
```

Puesto que la columna Address de la tabla con tipo Person contiene tipos estructurados, tendrá que definir funciones adicionales y emitir SQL adicional para devolver los datos de dicha columna. Suponiendo que se realicen estos pasos adicionales, la consulta anterior devuelve la salida siguiente, donde *Additional Attributes* incluye GPA y Salary:

```
1                OID          NAME                Additional Attributes
---------------- ------------ ------------------- ...
PERSON_T         a            Andrew              ...
PERSON_T         b            Bob                 ...
PERSON_T         c            Cathy               ...
EMPLOYEE_T       d            Dennis              ...
EMPLOYEE_T       e            Eva                 ...
EMPLOYEE_T       f            Franky              ...
MANAGER_T        i            Iris                ...
ARCHITECT_T      l            Leo                 ...
EMPLOYEE_T       s            Susan               ...
```

Conceptos relacionados:

- "Tipos estructurados definidos por el usuario" en la página 266
- "Tablas con tipo" en la página 276
- "Vistas con tipo" en la página 290

Tareas relacionadas:

- "Almacenamiento de objetos de tipo estructurado en columnas de tablas" en la página 298
- "Emisión de consultas para deshacer referencias" en la página 294

Tipos estructurados como tipos de columna

Almacenamiento de objetos de tipo estructurado en columnas de tablas

El almacenamiento de objetos en columnas es de utilidad cuando existe la necesidad de modelar hechos sobre objetos del negocio que no se pueden modelar adecuadamente con los tipos de datos incorporados en DB2. En otras palabras, puede almacenar los objetos del negocio (como, por ejemplo, empleados, departamentos, etc.) en tablas con tipo, pero estos objetos también pueden tener atributos que se modelen mejor utilizando un tipo estructurado.

Por ejemplo, suponga que la aplicación necesita acceder a determinadas partes de una dirección. En lugar de almacenar la dirección en forma de caracteres desestructurados, la puede almacenar como un objeto estructurado, tal como se muestra en la Figura 11 en la página 299.

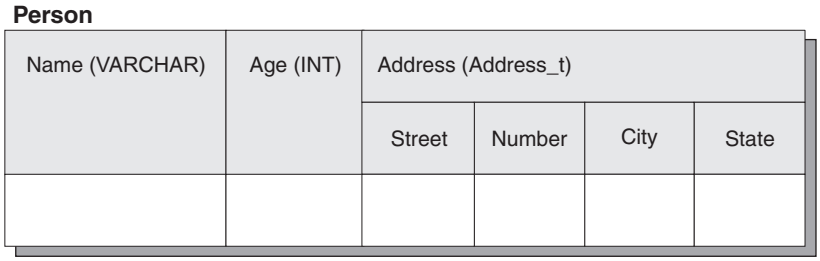


Figura 11. Atributo de dirección en forma de tipo estructurado

Además, puede definir una jerarquía de tipos de direcciones para modelar distintos formatos de direcciones que se utilicen en distintos países. Por ejemplo, es posible que desee incluir tanto un tipo de dirección estadounidense, que contiene un código postal, y un tipo de dirección brasileño, para el cual se requiere el atributo de barrio.

La Figura 12 muestra una jerarquía de los distintos tipos de direcciones. El tipo raíz es `Address_t`, que tiene tres subtipos, cada uno de los cuales tiene un atributo adicional que refleja algún aspecto de cómo se forman las direcciones en esa región.

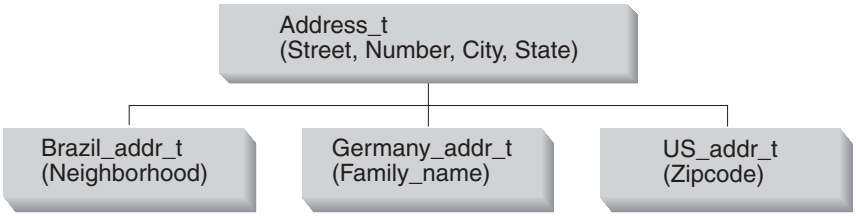


Figura 12. Jerarquía de tipos estructurados para el tipo Address_t

```
CREATE TYPE Address_t AS
   (street VARCHAR(30),
   number CHAR(15),
   city VARCHAR(30),
   state VARCHAR(10))
   MODE DB2SQL;

CREATE TYPE Germany_addr_t UNDER Address_t AS
   (family_name VARCHAR(30))
   MODE DB2SQL;

CREATE TYPE Brazil_addr_t UNDER Address_t AS
   (neighborhood VARCHAR(30))
   MODE DB2SQL;

CREATE TYPE US_addr_t UNDER Address_t AS
   (zip CHAR(10))
   MODE DB2SQL;
```

Si los objetos se almacenan en forma de valores de columna, los atributos no se representan externamente tal como se hace con los objetos almacenados en filas de tablas. En lugar de esto, deberá utilizar métodos para manipular sus atributos. DB2 genera métodos de *observador* para devolver atributos y métodos de *mutador* para cambiar atributos. En el ejemplo siguiente se utilizan, para cambiar una dirección, un método de observador y dos métodos de mutador, uno para el atributo `Number` y otro para el atributo `Street`:

```
UPDATE Employee
   SET Address=Address..Number('4869')..Street('Appletree')
   WHERE Name='Franky'
   AND Address..State='CA';
```

En el ejemplo anterior, la cláusula SET de la sentencia UPDATE invoca los métodos de mutador `Number` y `Street` para actualizar los atributos de las instancias de tipo `Address_t`.

Para posibilitar la actualización de instancias de tipos estructurados más complejas, especialmente las anidadas, DB2 también le permite profundizar hasta el atributo que se debe actualizar en el lado izquierdo de la cláusula SET:

```
UPDATE Employee
   SET Address..Number = '4869',
       Address..Street = 'Appletree'
   WHERE Name='Franky' AND Address..State='CA'
```

La cláusula WHERE restringe la operación de la sentencia de actualización mediante dos predicados: una comparación de igualdad para la columna `Name` y una comparación de igualdad que invoca al método de observador `State` de la columna `Address`.

Conceptos relacionados:

- "Tipos estructurados definidos por el usuario" en la página 266

Tareas relacionadas:

- "Creación de tipos estructurados" en la página 266
- "Almacenamiento de instancias de tipos estructurados" en la página 267
- "Inserción en columnas de atributos de tipo estructurado" en la página 300
- "Recuperación y modificación de valores de tipo estructurado en las columnas" en la página 303

Información relacionada:

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

Inserción en columnas de atributos de tipo estructurado

Para insertar un atributo de un tipo estructurado definido por el usuario en una columna que es del mismo tipo que el atributo utilizando SQL estático incorporado, encierre entre paréntesis la variable del lenguaje principal que representa a la instancia del tipo y añada el operador de dos puntos y el nombre de atributo al paréntesis derecho. Por ejemplo, considere la situación siguiente:

```
- PERSON_T es un tipo estructurado que incluye el atributo NAME
     de tipo VARCHAR(30).
- T1 es una tabla que incluye una columna C1 de tipo VARCHAR(30).
- personhv es la variable de lenguaje principal declarada para el
     tipo PERSON_T en el lenguaje de programación.
```

La sintaxis conveniente para insertar el atributo NAME en la columna C1 es:

```
EXEC SQL INSERT INTO T1 (C1) VALUES ((:personhv)..NAME)
```

Conceptos relacionados:

- "Métodos de observador para tipos estructurados" en la página 275

Tareas relacionadas:

- "Creación de tipos estructurados" en la página 266
- "Almacenamiento de objetos de tipo estructurado en columnas de tablas" en la página 298
- "Recuperación de atributos de tipo estructurado" en la página 304

Definición y modificación de tablas con columnas de tipo estructurado

La creación de una tabla con columnas de tipos estructurados no es, en su mayor parte, distinta de la creación de tablas que sólo contienen los tipos de datos SQL de DB2. A cada columna que se define se le asigna un tipo de datos correspondiente. Para las columnas de tipo estructurado, se proporciona el nombre del tipo estructurado como tipo de datos correspondiente. Por ejemplo, la sentencia ALTER TABLE siguiente añade una columna de tipo `Address_t` a una tabla sin tipo `Customer_List`:

```
ALTER TABLE Customer_List
   ADD COLUMN Address Address_t;
```

Ahora, las instancias de `Address_t` o de cualquiera de los subtipos de `Address_t` se pueden almacenar en esta tabla.

Si le preocupa cómo se disponen los tipos estructurados en el registro de datos, puede utilizar la cláusula INLINE LENGTH en la sentencia CREATE TYPE. Esta cláusula indicará el tamaño máximo de una instancia de un tipo estructurado en una columna. Si el tamaño de una instancia de un tipo estructurado es menor que el máximo definido, los datos se almacenarán en línea con el resto de los valores de la fila. Si el tamaño del tipo estructurado supera el máximo definido, los datos del tipo estructurado se almacenarán fuera de la tabla (muy probablemente en LOB).

Para acomodar cambios que efectúe en un tipo estructurado, puede modificar el tamaño de la columna de tipo estructurado afectada emitiendo la sentencia ALTER TABLE ALTER COLUMN SET INLINE LENGTH. Después de modificar la longitud de una columna, debe invocar al programa de utilidad REORG.

Conceptos relacionados:

- "Tipos estructurados definidos por el usuario" en la página 266

Tareas relacionadas:

- "Creación de tipos estructurados" en la página 266
- "Almacenamiento de objetos de tipo estructurado en columnas de tablas" en la página 298

Información relacionada:

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

Definición de tipos con atributos de tipo estructurado

Un tipo se puede crear con un atributo de tipo estructurado o se puede modificar (antes de utilizarlo) para añadir o eliminar un atributo de esta clase. Por ejemplo, la sentencia CREATE TYPE siguiente contiene un atributo de tipo `Address_t`:

```
CREATE TYPE Person_t AS
   (Name VARCHAR(20),
   Age INT,
   Address Address_t)
   REF USING VARCHAR(13)
   MODE DB2SQL;
```

Person_t se puede utilizar como tipo de una tabla, como tipo de una columna de una tabla normal o como atributo de otro tipo estructurado.

Tareas relacionadas:

- "Creación de tipos estructurados" en la página 266
- "Almacenamiento de objetos de tipo estructurado en columnas de tablas" en la página 298

Información relacionada:

- "Sentencia CREATE TYPE (Estructurado)" en la publicación *Consulta de SQL, Volumen 2*

Inserción de filas que contienen valores de tipo estructurado

Cuando se crea un tipo estructurado, DB2 genera automáticamente un método de constructor para el tipo, y genera métodos de mutador y observador para los atributos del tipo. Puede utilizar estos métodos para crear instancias de tipos estructurados e insertar dichas instancias en una columna de una tabla.

Supongamos que desea añadir una nueva fila a la tabla con tipo Employee y que quiere que dicha fila contenga una dirección. Al igual que con los tipos de datos incorporados, puede añadir esta fila utilizando INSERT con la cláusula VALUES. Sin embargo, cuando especifique el valor que se debe insertar en la dirección tendrá que invocar la función de constructor proporcionada por el sistema para crear el valor:

```
INSERT INTO Employee (Oid, Name, Age, SerialNum, Salary, Dept, Address)
   VALUES(Employee_t('m'), 'Marie', 35, 005, 55000, BusinessUnit_t(2),
   US_addr_t ( ) 1
      ..street('Bakely Avenue') 2
      ..number('555') 3
      ..city('San Jose') 4
      ..state('CA') 5
      ..zip('95141')); 6
```

La sentencia anterior crea una instancia del tipo US_addr_t realizando las tareas siguientes:

1. La llamada a US_addr_t() invoca la función de constructor para el tipo US_addr_t, a fin de crear una instancia del tipo con todos los atributos establecidos con valores nulos.
2. La llamada a ..street('Bakely Avenue') invoca al método de mutador para el atributo street, a fin de establecer su valor en 'Bakely Avenue'.
3. La llamada a ..number('555') invoca al método de mutador para el atributo number, a fin de establecer su valor en '555'.
4. La llamada a ..city('San Jose') invoca al método de mutador para el atributo city, a fin de establecer su valor en 'San Jose'.
5. La llamada a ..state('CA') invoca al método de mutador para el atributo state, a fin de establecer su valor en 'CA'.

6. La llamada `..zip('95141')` invoca al método de mutador para el atributo `zip`, a fin de establecer su valor en '95141'.

Observe que, aunque el tipo de la columna `Address` de la tabla `Employee` está definido como `Address_t`, la propiedad de posibilidad de sustitución significa que la puede llenar con una instancia de `US_addr_t`, puesto que `US_addr_t` es un subtipo de `Address_t`.

Para evitar el tener que llamar explícitamente a los métodos de mutador para cada atributo de un tipo estructurado cada vez que cree una instancia del tipo, considere la posibilidad de definir su propia función de constructor incorporada al SQL, que inicializará todos los atributos. El ejemplo siguiente contiene la declaración para una función de constructor incorporada al SQL para el tipo `US_addr_t`:

```
CREATE FUNCTION US_addr_t
   (street VARCHAR(30),
   number CHAR(15),
   city VARCHAR(30),
      state VARCHAR(20),
      zip CHAR(10))
   RETURNS US_addr_t
   LANGUAGE SQL
   RETURN US_addr_t()..street(street)..number(number)
       ..city(city)..state(state)..zip(zipcode);
```

El ejemplo siguiente muestra cómo crear una instancia del tipo US_addr_t llamando a la función de constructor incorporada al SQL del ejemplo anterior:

```
INSERT INTO Employee(Oid, Name, Age, SerialNum, Salary, Dept, Address)
   VALUES(Employee_t('m'), 'Marie', 35, 005, 55000, BusinessUnit_t(2),
      US_addr_t('Bakely Avenue', '555', 'San Jose', 'CA', '95141'));
```

Conceptos relacionados:

- “Posibilidad de sustitución en las tablas con tipo” en la página 280
- “Tablas con tipo” en la página 276

Tareas relacionadas:

- “Creación de tipos estructurados” en la página 266
- “Almacenamiento de objetos de tipo estructurado en columnas de tablas” en la página 298
- “Inserción en columnas de atributos de tipo estructurado” en la página 300
- “Definición y modificación de tablas con columnas de tipo estructurado” en la página 301
- “Definición de tipos con atributos de tipo estructurado” en la página 301

Modificación de valores de tipo estructurado en columnas

Recuperación y modificación de valores de tipo estructurado en las columnas

Existen dos maneras de que las aplicaciones y las funciones definidas por el usuario puedan acceder a datos de columnas de tipo estructurado: accediendo a atributos individuales de un objeto o tomando el objeto como un valor simple. Si desea tratar un objeto como un valor simple, antes tiene que definir funciones de transformación. Una vez que defina las funciones de transformación correctas, podrá seleccionar un objeto estructurado como cualquier otro valor:

```
SELECT Name, Dept, Address
   FROM Employee
   WHERE Salary > 20000;
```

Los temas siguientes describen cómo se puede acceder explícitamente a atributos individuales de un objeto invocando los métodos de observador y de mutador incorporados en DB2. Estos métodos incorporados no requieren que se defina una función de transformación.

Procedimiento:

1. Recuperación de atributos de tipo estructurado
2. Acceso a los atributos de subtipos
3. Modificación de atributos de tipo estructurado
4. Devolución de información sobre un tipo estructurado

Conceptos relacionados:

- "Funciones de transformación y grupos de transformación" en la página 306

Tareas relacionadas:

- "Recuperación de atributos de tipo estructurado" en la página 304
- "Acceso a los atributos de subtipos" en la página 305
- "Modificación de atributos de tipo estructurado" en la página 305
- "Devolución de información sobre un tipo estructurado" en la página 306
- "Almacenamiento de objetos de tipo estructurado en columnas de tablas" en la página 298
- "Inserción en columnas de atributos de tipo estructurado" en la página 300
- "Inserción de filas que contienen valores de tipo estructurado" en la página 302

Recuperación de atributos de tipo estructurado

Para acceder explícitamente a atributos individuales de un objeto, invoque los métodos de *observador* incorporados en DB2 sobre dichos atributos. Utilizando los métodos de observador, puede recuperar los atributos individualmente, en lugar de tratar el objeto como un solo valor.

El ejemplo siguiente accede a datos de la columna `Address` invocando los métodos de observador sobre `Address_t`, el tipo estático definido para la columna `Address`:

```
SELECT Name, Dept, Address..street, Address..number, Address..city,
   Address..state
   FROM Employee
   WHERE Salary > 20000;
```

Nota: DB2 permite invocar métodos que no tienen parámetros utilizando *<nombre-tipo>..<nombre-método>*() o *<nombre-tipo>..<nombre-método>*, donde *nombre-tipo* representa el nombre del tipo estructurado y *nombre-atributo* representa el nombre del método que no tiene parámetros.

También puede utilizar métodos de observador para seleccionar cada uno de los atributos en una variable del lenguaje principal, del modo siguiente:

```
SELECT Name, Dept, Address..street, Address..number, Address..city,
   Address..state
   INTO :name, :dept, :street, :number, :city, :state
   FROM Employee
   WHERE Empno = '000250';
```

Tareas relacionadas:

- "Inserción en columnas de atributos de tipo estructurado" en la página 300
- "Acceso a los atributos de subtipos" en la página 305
- "Modificación de atributos de tipo estructurado" en la página 305
- "Devolución de información sobre un tipo estructurado" en la página 306

Acceso a los atributos de subtipos

En la tabla `Employee`, las direcciones pueden ser de 4 tipos diferentes: `Address_t`, `US_addr_t`, `Brazil_addr_t` y `Germany_addr_t`. Para acceder a los atributos de valores de uno de los subtipos de `Address_t`, debe utilizar la expresión TREAT para indicar a DB2 que un objeto determinado puede ser de los tipos `US_addr_t`, `Germany_addr_t` o `Brazil_addr_t`. La expresión TREAT convierte el tipo de datos de un tipo estructurado en uno de sus subtipos, tal como se muestra en la consulta siguiente:

```
SELECT Name, Dept, Address..street, Address..number, Address..city,
   Address..state,
   CASE
      WHEN Address IS OF (US_addr_t)
      THEN TREAT(Address AS US_addr_t)..zip
      WHEN Address IS OF (Germany_addr_t)
      THEN TREAT (Address AS Germany_addr_t)..family_name
      WHEN Address IS OF (Brazil_addr_t)
      THEN TREAT (Address AS Brazil_addr_t)..neighborhood
   ELSE NULL END
   FROM Employee
   WHERE Salary > 20000;
```

Tareas relacionadas:

- "Inserción en columnas de atributos de tipo estructurado" en la página 300
- "Recuperación de atributos de tipo estructurado" en la página 304
- "Modificación de atributos de tipo estructurado" en la página 305
- "Devolución de información sobre un tipo estructurado" en la página 306

Modificación de atributos de tipo estructurado

Para cambiar un atributo del valor de una columna estructurada, invoque al método de mutador para el atributo que desea cambiar. Por ejemplo, para cambiar el atributo `street` de una dirección, puede invocar al método de mutador para `street` con el valor por el que se cambiará. El valor devuelto es una dirección con el nuevo valor para `street`. El ejemplo siguiente invoca un método de mutador para el atributo denominado `street` a fin de actualizar un tipo de dirección en la tabla `Employee`:

```
UPDATE Employee
   SET Address = Address..street('Bailey')
   WHERE Address..street = 'Bakely';
```

El ejemplo siguiente realiza la misma actualización que el ejemplo anterior, pero en lugar de mencionar la columna estructurada para la actualización, la cláusula SET accede directamente al método de mutador para el atributo mencionado, `street`:

```
UPDATE Employee
   SET Address..street = 'Bailey'
   WHERE Address..street = 'Bakely';
```

Tareas relacionadas:

- "Inserción en columnas de atributos de tipo estructurado" en la página 300

- "Recuperación de atributos de tipo estructurado" en la página 304
- "Acceso a los atributos de subtipos" en la página 305
- "Devolución de información sobre un tipo estructurado" en la página 306

Devolución de información sobre un tipo estructurado

Puede utilizar funciones incorporadas para devolver el nombre, el esquema o el ID de tipo interno de un tipo determinado. La sentencia siguiente devuelve el tipo exacto del valor de dirección asociado al empleado llamado 'Iris':

```
SELECT TYPE_NAME(Address)
   FROM Employee
   WHERE Name='Iris';
```

Tareas relacionadas:

- "Inserción en columnas de atributos de tipo estructurado" en la página 300
- "Recuperación de atributos de tipo estructurado" en la página 304
- "Acceso a los atributos de subtipos" en la página 305
- "Modificación de atributos de tipo estructurado" en la página 305

Funciones de transformación y grupos de transformación

Funciones de transformación y grupos de transformación

Las *funciones de transformación* se utilizan para intercambiar valores de tipo estructurado con programas del lenguaje principal y con funciones y métodos externos. Las funciones de transformación se producen, naturalmente, por pares: una función de transformación FROM SQL y una función de transformación TO SQL. La función FROM SQL convierte un objeto de tipo estructurado en un tipo que se pueda intercambiar con un programa externo, y la función TO SQL construye el objeto.

Cuando se crean funciones de transformación, se pone cada par lógico de funciones de transformación en un grupo. El nombre del *grupo de transformación* identifica de forma exclusiva a un par de estas funciones para un tipo estructurado indicado.

Para poder utilizar una función de transformación, antes debe usar la sentencia CREATE TRANSFORM para asociar la función de transformación a un nombre de grupo y a un tipo. La sentencia CREATE TRANSFORM identifica una o más funciones existentes y hace que se utilicen como funciones de transformación. El ejemplo siguiente denomina dos pares de funciones que se utilizarán como funciones de transformación para el tipo `Address_t`. La sentencia crea dos grupos de transformación, `func_group` y `client_group`, cada uno de los cuales consta de una transformación FROM SQL y una transformación TO SQL.

```
CREATE TRANSFORM FOR Address_t
   func_group ( FROM SQL WITH FUNCTION addresstofunc,
      TO SQL WITH FUNCTION functoaddress )
   client_group ( FROM SQL WITH FUNCTION stream_to_client,
      TO SQL WITH FUNCTION stream_from_client ) ;
```

Puede asociar funciones adicionales al tipo `Address_t` añadiendo más grupos en la sentencia CREATE TRANSFORM. Para modificar la definición de la transformación, debe volver a emitir la sentencia CREATE TRANSFORM con las funciones adicionales.

Utilice la sentencia DROP TRANSFORM de SQL para desasociar las funciones de transformación de los tipos. Después de ejecutar la sentencia DROP TRANSFORM, las funciones seguirán existiendo pero ya no se utilizarán como funciones de transformación para este tipo. El ejemplo siguiente desasocia el grupo de funciones de transformación específico `func_group` para el tipo `Address_t`, y luego desasocia todas las funciones de transformación para el tipo `Address_t`:

```
DROP TRANSFORMS func_group FOR Address_t;

DROP TRANSFORMS ALL FOR Address_t;
```

Para modificar la definición de la transformación, debe volver a emitir la sentencia CREATE TRANSFORM con las funciones adicionales. Por ejemplo, puede ser conveniente personalizar las funciones de cliente para distintos programas del lenguaje principal, como, por ejemplo, disponer de una para C y otra para Java™. Para optimizar el rendimiento de la aplicación, puede que desee que las transformaciones sólo actúen con un subconjunto de los atributos de los objetos. O es posible que desee tener una transformación que utilice VARCHAR como representación del cliente para un objeto y una transformación que utilice BLOB.

Conceptos relacionados:

- "Tipos estructurados definidos por el usuario" en la página 266
- "Requisitos de las funciones de transformación" en la página 321
- "Especificación de grupos de transformación" en la página 308
- "Correlaciones del programa del lenguaje principal con funciones de transformación" en la página 310
- "Transformaciones de función" en la página 311
- "Recomendaciones para denominar grupos de transformación" en la página 307

Tareas relacionadas:

- "Recuperación y modificación de valores de tipo estructurado en las columnas" en la página 303

Información relacionada:

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

Recomendaciones para denominar grupos de transformación

Los nombres de grupo de transformación son identificadores *no calificados*; es decir, no tienen asociado un esquema específico. A no ser que escriba transformaciones para manejar parámetros de subtipos, no debe asignar un nombre de grupo de transformación distinto para cada tipo estructurado. Puesto que es posible que necesite utilizar varios tipos no relacionados distintos en el mismo programa o en la misma sentencia de SQL, debe denominar los grupos de transformación en función de las tareas realizadas por las funciones de transformación.

En general, los nombres de los grupos de transformación deben reflejar las funciones que realizan sin confiar en los nombres de tipo ni reflejar de ningún modo la lógica de las funciones de transformación, que probablemente serán muy diferentes en los distintos tipos. Por ejemplo, podría utilizar el nombre `func_group` o `object_functions` para cualquier grupo en que estén definidas las transformaciones de función TO y FROM SQL. Podría utilizar el nombre `client_group` o `program_group` para un grupo que contenga transformaciones de clientes TO y FROM SQL.

En el ejemplo siguiente, los tipos `Address_t` y `Polygon` utilizan transformaciones muy diferentes, pero usan los mismos nombres de grupo de funciones.

```
CREATE TRANSFORM FOR Address_t
   func_group (TO SQL WITH FUNCTION functoaddress,
   FROM SQL WITH FUNCTION addresstofunc );

CREATE TRANSFORM FOR Polygon
   func_group (TO SQL WITH FUNCTION functopolygon,
   FROM SQL WITH FUNCTION polygontofunc);
```

Una vez que se establece el grupo de transformación en `func_group` en la situación apropiada, DB2® invoca la función de transformación correcta siempre que se enlaza o se desenlaza una dirección o un polígono.

Restricción: No se puede empezar un grupo de transformación por la serie 'SYS'; este grupo está reservado para uso de DB2.

Cuando se define una función o un método externos y no se especifica un nombre de grupo de transformación, DB2 intenta utilizar el nombre DB2_FUNCTION y supone que se ha especificado este nombre de grupo para el tipo estructurado indicado. Si no se especifica un nombre de grupo al precompilar un programa cliente que hace referencia a un tipo estructurado indicado, DB2 intenta utilizar un nombre de grupo denominado DB2_PROGRAM y, nuevamente, supone que se ha definido este nombre de grupo para ese tipo.

Este comportamiento por omisión resulta conveniente en algunos casos pero, en un esquema de bases de datos más complejo, es posible que desee un convenio ligeramente más general para los nombres de los grupos de transformación. Por ejemplo, le puede ser de ayuda utilizar nombres de grupo distintos correspondientes a los lenguajes distintos para los que se puede desenlazar el tipo.

Conceptos relacionados:

- "Funciones de transformación y grupos de transformación" en la página 306
- "Especificación de grupos de transformación" en la página 308

Información relacionada:

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

Especificación de grupos de transformación

Especificación de grupos de transformación

Se pueden definir muchos grupos de transformación para un tipo estructurado determinado, por lo que debe especificar qué grupo de transformaciones se debe utilizar para dicho tipo en un programa o en una sentencia de SQL específicos. Existen tres circunstancias en las que debe especificar grupos de transformación:

- Cuando defina una función o un método externos, debe especificar el grupo que *descompone* y *construye* un objeto referenciado.
- Cuando precompile o enlace SQL estático, debe especificar el grupo de transformaciones que realizan el enlace y el desenlace del cliente para un tipo referenciado.
- Cuando ejecute SQL dinámico, o cuando utilice el Procesador de línea de mandatos, debe especificar el grupo de transformaciones que realizan el enlace y el desenlace del cliente para un tipo referenciado.

Conceptos relacionados:

- "Funciones de transformación y grupos de transformación" en la página 306
- "Correlaciones del programa del lenguaje principal con funciones de transformación" en la página 310

Tareas relacionadas:

- "Especificación de grupos de transformación para rutinas externas" en la página 309
- "Especificación de grupos de transformación para SQL dinámico" en la página 309
- "Especificación de grupos de transformación para SQL estático" en la página 310

Especificación de grupos de transformación para rutinas externas

Las sentencias CREATE FUNCTION y CREATE METHOD permiten especificar la cláusula TRANSFORM GROUP, que sólo es válida si el valor de la cláusula LANGUAGE no es SQL. Las funciones del lenguaje SQL no precisan de transformaciones, mientras que las funciones externas las necesitan. La cláusula TRANSFORM GROUP permite especificar, para cualquier función o método indicados, el grupo de transformación que contiene las transformaciones TO SQL y FROM SQL utilizadas para los parámetros y resultados de tipos estructurados. En el ejemplo siguiente, las sentencias CREATE FUNCTION y CREATE METHOD especifican el grupo de transformación `func_group` para las transformaciones TO SQL y FROM SQL:

```
CREATE FUNCTION stream_from_client (VARCHAR (150))
   RETURNS Address_t
   ...
   TRANSFORM GROUP func_group
   EXTERNAL NAME 'addressudf!address_stream_from_client'
   ...

CREATE METHOD distance ( point )
   FOR polygon
   RETURNS integer
   :
   TRANSFORM GROUP func_group ;
```

Conceptos relacionados:

- "Funciones de transformación y grupos de transformación" en la página 306
- "Especificación de grupos de transformación" en la página 308

Tareas relacionadas:

- "Definición del comportamiento de los tipos estructurados" en la página 271
- "Especificación de grupos de transformación para SQL dinámico" en la página 309
- "Especificación de grupos de transformación para SQL estático" en la página 310

Especificación de grupos de transformación para SQL dinámico

Si utiliza SQL dinámico, puede establecer el registro especial CURRENT DEFAULT TRANSFORM GROUP. Este registro especial no se utiliza para las sentencias de SQL estático ni para el intercambio de parámetros y resultados con funciones externas o métodos. Utilice la sentencia SET CURRENT DEFAULT TRANSFORM GROUP para establecer el grupo de transformación por omisión para las sentencias de SQL dinámico:

```
SET CURRENT DEFAULT TRANSFORM GROUP = client_group;
```

Conceptos relacionados:

- "Funciones de transformación y grupos de transformación" en la página 306
- "Especificación de grupos de transformación" en la página 308

Tareas relacionadas:

- "Especificación de grupos de transformación para rutinas externas" en la página 309
- "Especificación de grupos de transformación para SQL estático" en la página 310

Especificación de grupos de transformación para SQL estático

Para el SQL estático, utilice la opción TRANSFORM GROUP en los mandatos PRECOMPILE o BIND para especificar el grupo de transformación estático utilizado por las sentencias de SQL estático para intercambiar valores de diversos tipos con los programas del sistema principal. Los grupos de transformación estáticos no se aplican a las sentencias de SQL dinámico ni al intercambio de parámetros y resultados con funciones externas o métodos. Para especificar el grupo de transformación estático en los mandatos PRECOMPILE o BIND, utilice la cláusula TRANSFORM GROUP:

```
PRECOMPILE ...
TRANSFORM GROUP client_group
... ;
```

Conceptos relacionados:

- "Funciones de transformación y grupos de transformación" en la página 306
- "Especificación de grupos de transformación" en la página 308

Tareas relacionadas:

- "Especificación de grupos de transformación para rutinas externas" en la página 309
- "Especificación de grupos de transformación para SQL dinámico" en la página 309

Información relacionada:

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

Creación de la correlación con el programa del lenguaje principal

Correlaciones del programa del lenguaje principal con funciones de transformación

Una aplicación no puede seleccionar directamente un objeto entero aunque se pueden seleccionar atributos individuales de un objeto en una aplicación. Normalmente, una aplicación no inserta directamente un objeto entero, aunque puede insertar el resultado de una invocación a la función de constructor:

```
INSERT INTO Employee(Address) VALUES (Address_t());
```

Para intercambiar objetos enteros entre las aplicaciones servidor y cliente, o funciones externas, normalmente deberá escribir *funciones de transformación*.

Una función de transformación define la manera en que DB2® convierte un objeto a un formato bien definido para acceder a su contenido, o *desenlaza* el objeto. Una función de transformación distinta define la manera en que DB2 devuelve el objeto que se almacenará en la base de datos, o *enlaza* el objeto. Las transformaciones que desenlazan un objeto se denominan funciones de transformación FROM SQL, y las transformaciones que enlazan un objeto de una columna se denominan transformaciones TO SQL.

Muy probablemente, existirán distintas transformaciones para pasar objetos a *rutinas*, o a UDF externas y métodos, diferentes de las utilizadas para pasar objetos a aplicaciones cliente. Esto es debido a que, cuando se pasa el objeto a una rutina externa, se *descompone* el objeto y se pasa a la rutina en forma de lista de parámetros. Con la aplicación cliente, se debe pasar el objeto a un tipo incorporado simple, como por ejemplo BLOB. Este proceso se denomina codificación del objeto. A menudo, estos dos tipos de transformaciones se utilizan juntos.

Utilice la sentencia CREATE TRANSFORM de SQL para asociar funciones de transformación a un tipo estructurado determinado. En la sentencia CREATE TRANSFORM, las funciones se emparejan en lo que se denominan *grupos de transformación*. Esto facilita la identificación de las funciones que se utilizan para un objetivo de transformación determinado. Cada grupo de transformación no puede contener más de una transformación FROM SQL, ni más de una transformación TO SQL, para un tipo determinado.

Conceptos relacionados:

- "Requisitos de las funciones de transformación" en la página 321
- "Funciones de transformación y grupos de transformación" en la página 306
- "Transformaciones de función" en la página 311
- "Transformaciones de cliente" en la página 316

Tareas relacionadas:

- "Implementación de transformaciones de función utilizando rutinas incorporadas al SQL" en la página 313
- "Pase de parámetros de tipo estructurado a rutinas externas" en la página 314

Información relacionada:

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

Transformaciones de función

DB2® utiliza las *transformaciones de función* TO SQL y FROM SQL para pasar un objeto a una rutina externa y viceversa. No existe ninguna necesidad de utilizar transformaciones para las rutinas incorporadas al SQL. No obstante, DB2 utiliza a menudo estas funciones formando parte del proceso de pasar un objeto a un programa cliente, y viceversa.

El ejemplo siguiente emite una sentencia de SQL que invoca una UDF externa llamada `MYUDF`, que toma una dirección como parámetro de entrada, modifica la dirección (por ejemplo, para reflejar un cambio en los nombres de calle) y devuelve la dirección modificada:

```
SELECT MYUDF(Address)
FROM PERSON;
```

La Figura 13 muestra cómo procesa DB2 la dirección.

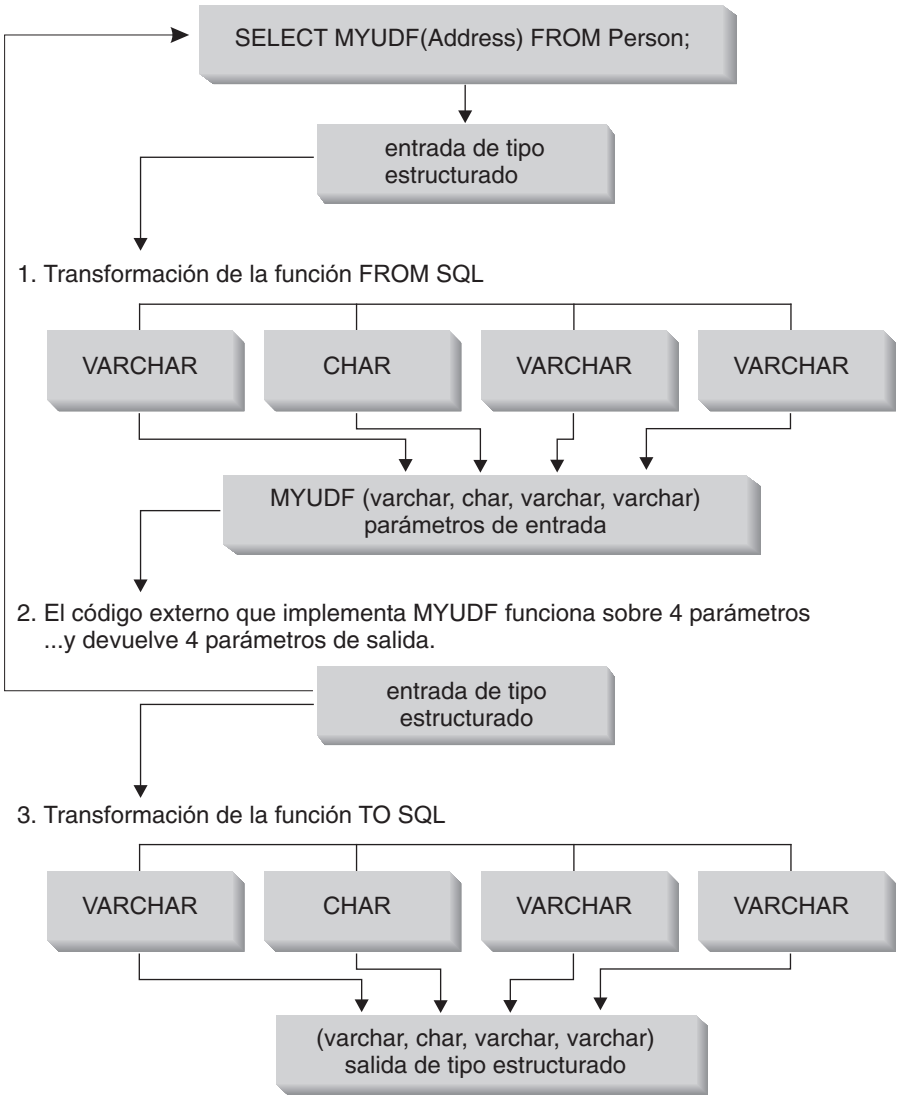


Figura 13. Intercambio de un parámetro de tipo estructurado con una rutina externa

1. La función de transformación FROM SQL descompone el objeto estructurado en un conjunto ordenado de sus atributos base. Esto permite que la rutina reciba el objeto como una simple lista de parámetros cuyos tipos son tipos de datos básicos incorporados. Por ejemplo, suponga que desea pasar un objeto de dirección a una rutina externa. Los atributos de `Address_t` son VARCHAR, CHAR, VARCHAR y VARCHAR, en este orden. La transformación FROM SQL para pasar este objeto a una rutina tiene que aceptar este objeto como entrada y devolver VARCHAR, CHAR, VARCHAR y VARCHAR. Luego, estas salidas se pasan a la rutina externa en forma de cuatro parámetros separados, con cuatro parámetros de indicador de nulo correspondientes y un indicador de nulo para el propio tipo estructurado. El orden de los parámetros en la función FROM SQL no importa, siempre que todas las funciones que devuelvan tipos `Address_t` utilicen el mismo orden.
2. La rutina externa acepta la dirección descompuesta como parámetros de entrada, realiza su proceso sobre estos valores y, a continuación, devuelve los atributos como parámetros de salida.

3. La función de transformación TO SQL debe volver a cambiar los parámetros VARCHAR, CHAR, VARCHAR y VARCHAR devueltos por `MYUDF` por un objeto de tipo `Address_t`. En otras palabras, la función de transformación TO SQL debe tomar los cuatro parámetros, y todos los parámetros de indicador de nulo correspondientes, como valores de salida de la rutina. La función TO SQL construye el objeto estructurado y luego muta los atributos con los valores indicados.

Nota: Si `MYUDF` devuelve también un tipo estructurado, otra función de transformación debe transformar el tipo estructurado resultante cuando se utilice la UDF en una cláusula SELECT. Para evitar la creación de otra función de transformación, puede utilizar sentencias SELECT con métodos de observador, tal como en el ejemplo siguiente:

```
SELECT Name
   FROM Employee
   WHERE MYUDF(Address)..city LIKE 'Tor%';
```

Conceptos relacionados:

- "Funciones de transformación y grupos de transformación" en la página 306
- "Correlaciones del programa del lenguaje principal con funciones de transformación" en la página 310
- "Transformaciones de cliente" en la página 316

Tareas relacionadas:

- "Implementación de transformaciones de función utilizando rutinas incorporadas al SQL" en la página 313
- "Pase de parámetros de tipo estructurado a rutinas externas" en la página 314

Implementación de transformaciones de función utilizando rutinas incorporadas al SQL

Para descomponer y construir objetos al intercambiar el objeto con una rutina externa, debe utilizar funciones definidas por el usuario escritas en SQL, llamadas rutinas incorporadas al SQL. Para crear una función incorporada al SQL, emita una sentencia CREATE FUNCTION con la cláusula LANGUAGE SQL.

En la función incorporada al SQL, puede utilizar constructores, observadores y mutadores para lograr la transformación. Esta transformación incorporada al SQL interviene entre la sentencia de SQL y la función externa. La transformación FROM SQL toma el objeto como parámetro de SQL y devuelve una fila de valores que representan los atributos del tipo estructurado. El ejemplo siguiente contiene una función de transformación FROM SQL posible para un objeto de dirección utilizando una función incorporada al SQL:

```
CREATE FUNCTION addresstofunc (A Address_t)  1
   RETURNS ROW  (Street VARCHAR(30), Number CHAR(15),
      City VARCHAR(30), State (VARCHAR(10))   2

   LANGUAGE SQL  3
   RETURN VALUES (A..Street, A..Number, A..City, A..State)  4
```

La lista siguiente explica la sintaxis de la sentencia CREATE FUNCTION anterior:

1. La signatura de esta función indica que acepta un parámetro, como objeto del tipo `Address_t`.

2. La cláusula RETURNS ROW indica que la función devuelve una fila que contiene cuatro columnas: `Street`, `Number`, `City` y `State`.
3. La cláusula LANGUAGE SQL indica que se trata de una función incorporada al SQL, y no de una función externa.
4. La cláusula RETURN marca el comienzo del cuerpo de la función. El cuerpo consta de una sola cláusula VALUES, que invoca al método de observador para cada atributo del objeto `Address_t`. Los métodos de observador descomponen el objeto en un conjunto de tipos base, que la función devuelve como una fila.

DB2 no sabe que se pretende utilizar esta función como función de transformación. Hasta que cree un grupo de transformación que utilice esta función, y después especifique dicho grupo de transformación en la situación apropiada, DB2 no podrá utilizar la función como función de transformación.

La transformación TO SQL realiza, simplemente, lo contrario que la función FROM SQL. Toma como entrada la lista de parámetros procedentes de una rutina y devuelve una instancia del tipo estructurado. Para construir el objeto, la función FROM SQL siguiente invoca la función de constructor para el tipo `Address_t`:

```
CREATE FUNCTION functoaddress (street VARCHAR(30), number CHAR(15),
                               city VARCHAR(30), state VARCHAR(10)) 1
   RETURNS Address_t 2
   LANGUAGE SQL
   CONTAINS SQL
   RETURN
      Address_t()..street(street)..number(number)
      ..city(city)..state(state) 3
```

La lista siguiente explica la sintaxis de la sentencia anterior:

1. La función toma un conjunto de atributos del tipo base.
2. La función devuelve un tipo estructurado `Address_t`.
3. La función construye el objeto a partir de los tipos de entrada, invocando al constructor para `Address_t` y a los mutadores para cada uno de los atributos.

El orden de los parámetros en la función FROM SQL no importa, aparte de que todas las funciones que devuelvan direcciones utilizando esta función de transformación deben utilizar el mismo orden.

Conceptos relacionados:

- "Transformaciones de función" en la página 311

Información relacionada:

- "Sentencia CREATE FUNCTION (Escalar de SQL, tabla o fila)" en la publicación *Consulta de SQL, Volumen 2*

Pase de parámetros de tipo estructurado a rutinas externas

Cuando pase parámetros de tipo estructurado a una rutina externa, debe pasar un parámetro para cada atributo. Debe pasar un indicador de nulo para cada parámetro y un indicador de nulo para el propio tipo estructurado. El ejemplo siguiente acepta el tipo estructurado `Address_t` y devuelve un tipo base:

```
CREATE FUNCTION stream_to_client (Address_t)
   RETURNS VARCHAR(150) ...
```

La rutina externa debe aceptar el indicador de nulo para la instancia del tipo `Address_t` (address_ind) y un indicador de nulo para cada uno de los atributos del

tipo Address_t. También hay un indicador de nulo para el parámetro de salida VARCHAR. El código siguiente representa las cabeceras de función en lenguaje C para las funciones que implementan las UDF:

```
void SQL_API_FN stream_to_client(
/* dirección descompuesta */
   SQLUDF_VARCHAR  *street,
   SQLUDF_CHAR     *number,
   SQLUDF_VARCHAR  *city,
   SQLUDF_VARCHAR  *state,
/* salida VARCHAR */
   SQLUDF_VARCHAR  *output,
/*indicadores de nulo para los atributos de tipo */
   SQLUDF_NULLIND *street_ind,
   SQLUDF_NULLIND  *number_ind,
   SQLUDF_NULLIND  *city_ind,
   SQLUDF_NULLIND  *state_ind,
/*indicador de nulo para la instancia del tipo */
   SQLUDF_NULLIND *address_ind,
/*indicador de nulo para la salida VARCHAR */
   SQLUDF_NULLIND *out_ind,
   SQLUDF_TRAIL_ARGS)
```

Suponga que la rutina acepta dos parámetros de tipo estructurado distintos, *st1* y *st2*, y devuelve otro tipo estructurado *st3*:

```
CREATE FUNCTION myudf (int, st1, st2)
   RETURNS st3
```

Tabla 35. Atributos de los parámetros de myudf

ST1	ST2	ST3
st1_att1 VARCHAR	st2_att1 VARCHAR	st3_att1 INTEGER
st2_att2 INTEGER	st2_att2 CHAR	st3_att2 CLOB
	st2_att3 INTEGER	

El código siguiente representa las cabeceras del lenguaje C para las rutinas que implementan las UDF. Los argumentos incluyen variables e indicadores de nulo para los atributos del tipo estructurado descompuesto, y un indicador de nulo para cada instancia de un tipo estructurado, del modo siguiente:

```
void SQL_API_FN myudf(
    SQLUDF_INTEGER *INT,
 /* Entrada st1 descompuesta */
   SQLUDF_VARCHAR *st1_att1,
   SQLUDF_INTEGER  *st1_att2,
 /* Entrada st2 descompuesta */
   SQLUDF_VARCHAR *st2_att1,
   SQLUDF_CHAR   *st2_att2,
   SQLUDF_INTEGER *st2_att3,
 /* Salida st3 descompuesta */
   SQLUDF_VARCHAR *st3_att1out,
   SQLUDF_CLOB   *st3_att2out,
 /* Indicador de nulo del entero */
   SQLUDF_NULLIND *INT_ind,
 /* Indicadores de nulo de atributos y tipo de st1 */
   SQLUDF_NULLIND *st1_att1_ind,
   SQLUDF_NULLIND *st1_att2_ind,
   SQLUDF_NULLIND *st1_ind,
 /* Indicadores de nulo de atributos y tipo de st2 */
   SQLUDF_NULLIND *st2_att1_ind,
   SQLUDF_NULLIND *st2_att2_ind,
   SQLUDF_NULLIND *st2_att3_ind,
   SQLUDF_NULLIND *st2_ind,
```

```
/* Indicadores de nulo de atributos de salida y tipo de st3 */
   SQLUDF_NULLIND *st3_att1_ind,
   SQLUDF_NULLIND *st3_att2_ind,
   SQLUDF_NULLIND *st3_ind,
/* argumentos de cola */
   SQLUDF_TRAIL_ARGS
)
```

Conceptos relacionados:

- "Funciones de transformación y grupos de transformación" en la página 306
- "Correlaciones del programa del lenguaje principal con funciones de transformación" en la página 310
- "Transformaciones de función" en la página 311
- "Transformaciones de cliente" en la página 316

Tareas relacionadas:

- "Implementación de transformaciones de cliente para enlazar desde un cliente mediante UDF externas" en la página 319

Transformaciones de cliente

Las *transformaciones de cliente* intercambian tipos estructurados con programas de aplicación clientes. Por ejemplo, suponga que desea ejecutar la sentencia de SQL siguiente:

```
...
SQL TYPE IS Address_t AS VARCHAR(150) addhv;
...

EXEC SQL SELECT Address
   FROM Person
   INTO :addhv
   WHERE AGE > 25
END EXEC;
```

La Figura 14 en la página 317 muestra el proceso de desenlazar esta dirección del programa cliente.

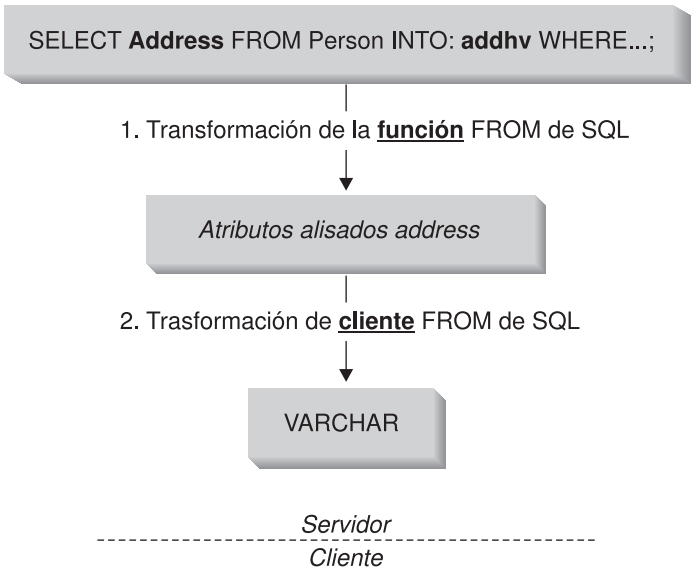


Figura 14. Desenlace de un tipo estructurado de una aplicación cliente

1. Primero hay que pasar el objeto a la transformación de función FROM SQL para descomponerlo en sus atributos tipo base propios.
2. La transformación de cliente FROM SQL debe codificar el valor en un solo tipo incorporado, como por ejemplo VARCHAR o BLOB. Este permite que el programa cliente reciba el valor entero en una única variable del lenguaje principal.

   Esta codificación puede ser tan simple como copiar los atributos a un área de almacenamiento contigua (proporcionando las alineaciones necesarias). Puesto que, generalmente, la codificación y decodificación de atributos no se puede lograr mediante el SQL, las transformaciones de cliente se suelen escribir en forma de UDF externas.
3. El programa cliente procesa el valor.

La Figura 15 en la página 318 muestra el proceso invertido de pasar la dirección a la base de datos.

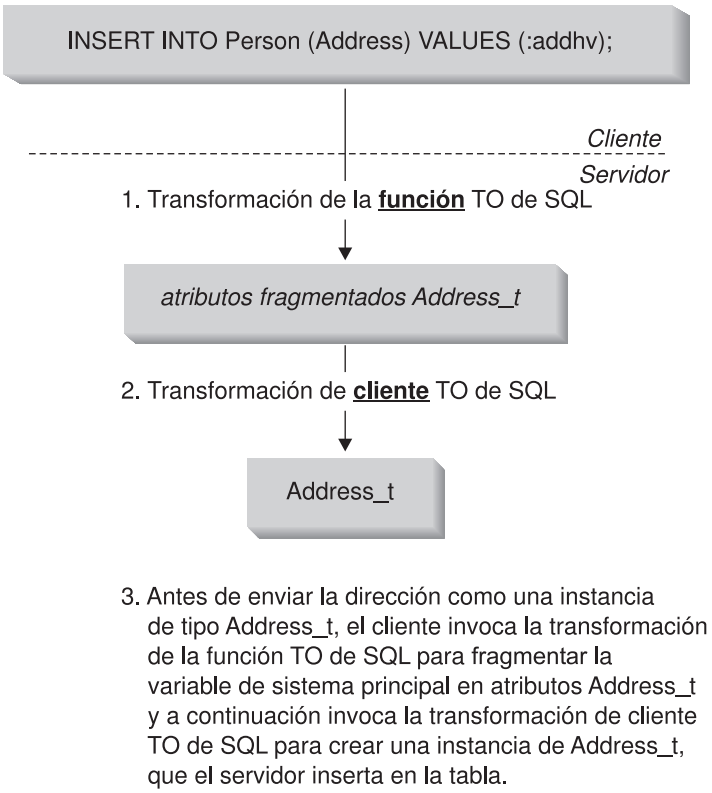


Figura 15. Enlace de un tipo estructurado desde un cliente

1. La aplicación cliente codifica la dirección en un formato esperado por la transformación de cliente TO SQL.
2. La transformación de cliente TO SQL descompone el único tipo incorporado en un conjunto de sus atributos de tipo base, que se utiliza como entrada para la transformación de la función TO SQL.
3. La transformación de función TO SQL construye la dirección y la devuelve a la base de datos.

Incluya la cláusula TRANSFORM GROUP para indicar a DB2® qué conjunto de transformaciones debe utilizar al procesar el tipo de dirección en la función indicada.

Conceptos relacionados:

- "Correlaciones del programa del lenguaje principal con funciones de transformación" en la página 310
- "Transformaciones de función" en la página 311

Tareas relacionadas:

- "Implementación de transformaciones de cliente mediante UDF externas" en la página 319
- "Implementación de transformaciones de cliente para enlazar desde un cliente mediante UDF externas" en la página 319

Implementación de transformaciones de cliente mediante UDF externas

Registre las transformaciones de cliente de la misma manera que cualquier otra UDF externa. Por ejemplo, suponga que ha escrito UDF externas que realizan la codificación y decodificación apropiadas para una dirección. Suponga que ha asignado a la transformación de cliente FROM SQL el nombre `from_sql_to_client` y a la transformación de cliente TO SQL el nombre `to_sql_from_client`. En ambos casos, la salida de las funciones está en un formato que las transformaciones de función FROM SQL y TO SQL apropiadas pueden utilizar como entrada.

```
CREATE FUNCTION from_sql_to_client (Address_t)
   RETURNS VARCHAR (150)
   LANGUAGE C
   TRANSFORM GROUP func_group
   EXTERNAL NAME 'addressudf!address_from_sql_to_client'
   NOT VARIANT
   NO EXTERNAL ACTION
   NOT FENCED
   NO SQL
   PARAMETER STYLE SQL;
```

El DDL del ejemplo anterior hace que parezca como si la UDF `from_sql_to_client` aceptara un parámetro del tipo `Address_t`. Lo que realmente sucede es que, para cada fila para la que se invoque la UDF `from_sql_to_client`, la transformación Addresstofunc descompone la `Address` en sus diversos atributos. La UDF `from_sql_to_client` produce una simple serie de caracteres y formatea los atributos de dirección para su visualización, con lo que permite utilizar la consulta de SQL simple siguiente para visualizar los atributos `Name` y `Address` para cada fila de la tabla `Person`:

```
SELECT Name, from_sql_to_client (Address)
FROM Person;
```

Observe que el DDL de `from_sql_to_client` incluye una cláusula llamada TRANSFORM GROUP. Esta cláusula indica a DB2 qué conjunto de transformaciones debe utilizar al procesar el tipo de direcciones en estas funciones.

Conceptos relacionados:

- "Transformaciones de cliente" en la página 316

Tareas relacionadas:

- "Pase de parámetros de tipo estructurado a rutinas externas" en la página 314
- "Implementación de transformaciones de cliente para enlazar desde un cliente mediante UDF externas" en la página 319

Implementación de transformaciones de cliente para enlazar desde un cliente mediante UDF externas

El DDL siguiente registra una función que toma el objeto codificado VARCHAR del cliente, lo descompone en sus diversos atributos de tipo base y lo pasa a la transformación de función TO SQL.

```
CREATE FUNCTION to_sql_from_client (VARCHAR (150))
   RETURNS Address_t
   LANGUAGE C
   TRANSFORM GROUP func_group
   EXTERNAL NAME 'addressudf!address_to_sql_from_client'
   NOT VARIANT
```

```
NO EXTERNAL ACTION
NOT FENCED
NO SQL
PARAMETER STYLE SQL;
```

Aunque parezca que `to_sql_from_client` devuelve directamente la dirección, lo que sucede realmente es que `to_sql_from_client` convierte el VARCHAR (150) en un conjunto de atributos de tipo base. A continuación, DB2 invoca implícitamente a la transformación TO SQL `functoaddress` para construir el objeto de dirección que se devuelve a la base de datos.

Observe que el DDL de `to_sql_from_client` incluye una cláusula llamada TRANSFORM GROUP. Esta cláusula indica a DB2 qué conjunto de transformaciones debe utilizar al procesar el tipo de direcciones en estas funciones.

Conceptos relacionados:

- "Transformaciones de cliente" en la página 316

Tareas relacionadas:

- "Implementación de transformaciones de cliente mediante UDF externas" en la página 319

Consideraciones sobre la conversión de datos

Cuando se intercambian datos, especialmente datos binarios, entre un servidor y un cliente, existen varios aspectos de la conversión de datos que se deben considerar. Por ejemplo, cuando se transfieren datos entre sistemas operativos que tienen esquemas de ordenación de bytes diferentes, los datos numéricos deben experimentar un proceso de inversión de bytes para restaurar su valor numérico correcto. Distintos sistemas operativos también tienen determinados requisitos de alineación para hacer referencia a datos numéricos contenidos en la memoria; algunos sistemas operativos ocasionarán excepciones de programa si no se observan estos requisitos. La base de datos convierte automáticamente los tipos de datos de caracteres, excepto si están incorporados en un tipo de datos binario, como por ejemplo BLOB o VARCHAR FOR BIT DATA.

Existen dos maneras de evitar problemas en la conversión de datos:

- Transforme siempre los objetos a tipos de datos de caracteres que se puedan imprimir, incluidos los datos numéricos.

  Este enfoque tiene las desventajas de ralentizar el rendimiento, debido a las muchas conversiones potenciales que se requieren, y de incrementar la complejidad del código que accede a estos objetos, por ejemplo en el cliente o en la propia función de transformación.
- Idee un formato neutro respecto al sistema operativo para un objeto transformado en un tipo de datos binario, similar al enfoque que toman las implementaciones de Java™. Asegúrese de:
  - Tener cuidado cuando empaquete o desempaquete estos objetos compactados, a fin de codificar o decodificar adecuadamente los tipos de datos individuales y para evitar que se corrompan los datos o se produzcan anomalías en el programa.
  - Incluir la información de cabecera suficiente en el tipo transformado, de forma que el resto del objeto codificado se pueda interpretar correctamente con independencia del sistema operativo del cliente o servidor.

– Utilizar la opción DBINFO de CREATE FUNCTION para pasar a la función de transformación diversas características relativas al entorno del servidor de bases de datos. Estas características se pueden incluir en la cabecera con un formato neutro respecto al sistema operativo.

Siempre que sea posible, debe escribir las funciones de transformación de forma que manejen correctamente todas las complejidades asociadas a la transferencia de datos entre servidor y cliente. Cuando diseñe una aplicación, tenga en cuenta los requisitos específicos del entorno y evalúe el equilibrio entre utilizar una generalización completa o una simplificación. Por ejemplo, si sabe que tanto el servidor de bases de datos como todos sus clientes trabajan en un entorno AIX® y utilizan la misma página de códigos, podría decidir ignorar las consideraciones tratadas previamente, puesto que actualmente no se requiere ninguna conversión. No obstante, si en un futuro cambia el entorno, es posible que tenga que ejercer un esfuerzo considerable para revisar el diseño original a fin de manejar correctamente la conversión de datos.

Conceptos relacionados:

- "Funciones de transformación y grupos de transformación" en la página 306
- "Correlaciones del programa del lenguaje principal con funciones de transformación" en la página 310
- "Transformaciones de función" en la página 311

Requisitos de las funciones de transformación

La Tabla 36 está destinada a serle de ayuda en la determinación de las funciones de transformación que necesita, según si está desenlazando desde una rutina externa o desde una aplicación cliente.

Tabla 36. Características de las funciones de transformación

Característica	**Intercambio de valores con una rutina externa**		**Intercambio de valores con una aplicación cliente**	
Dirección de la transformación	FROM SQL	TO SQL	FROM SQL	TO SQL
Qué se está transformando	Parámetro de rutina	Resultado de rutina	Variable del lenguaje principal de salida	Variable del lenguaje principal de entrada
Comportamiento	Descompone	Construye	Codifica	Decodifica
Parámetros de la función de transformación	Tipo estructurado	Fila de tipos incorporados	Tipo estructurado	Un tipo incorporado
Resultado de la función de transformación	Fila de tipos incorporados (probablemente atributos)	Tipo estructurado	Un tipo incorporado	Tipo estructurado
¿Depende de otra transformación?	No	No	Transformación de UDF FROM SQL	Transformación de UDF TO SQL
¿Cuándo se especifica el grupo de transformación?	Al registrar la UDF		Estática: durante la precompilación Dinámica: Registro especial	

Tabla 36. Características de las funciones de transformación (continuación)

Característica	Intercambio de valores con una rutina externa	Intercambio de valores con una aplicación cliente
¿Existen consideraciones sobre la conversión de datos?	No	Sí

Nota: Aunque generalmente no es el caso, las transformaciones de tipo cliente se pueden escribir realmente en SQL si se cumple alguna de las condiciones siguientes:

- El tipo estructurado sólo contiene un atributo.
- La codificación y decodificación de los atributos de un tipo incorporado se puede lograr mediante alguna combinación de funciones u operadores de SQL.

En estos casos, no es necesario depender de las transformaciones de función para intercambiar los valores de un tipo estructurado con una aplicación cliente.

Conceptos relacionados:

- "Funciones de transformación y grupos de transformación" en la página 306

Tareas relacionadas:

- "Recuperación de datos de subtipo de DB2" en la página 322

Recuperación de datos de subtipo de DB2

Si el modelo de datos se aprovecha de subtipos, el valor de una columna puede ser de muchos subtipos diferentes. Puede elegir dinámicamente las funciones de transformación correctas en base al tipo de entrada real.

Suponga que desea emitir la sentencia SELECT siguiente:

```
SELECT Address
   FROM Person
   INTO :hvaddr;
```

La aplicación no tiene manera de saber si se devolverá una instancia de `Address_t`, `US_addr_t`, etc. Para evitar que el ejemplo resulte demasiado complejo, suponga que sólo se puede devolver `Address_t` o `US_addr_t`. Las estructuras de estos tipos son distintas, por lo que las transformaciones que descomponen los atributos tienen que ser diferentes. Para asegurarse de que se invoque las transformaciones adecuadas:

Paso 1. Cree una transformación de función FROM SQL para cada variación de la dirección:

```
CREATE FUNCTION addresstofunc(A address_t)
   RETURNS ROW
   (Street VARCHAR(30), Number CHAR(15), City
   VARCHAR(30), STATE VARCHAR (10))
   LANGUAGE SQL
   RETURN VALUES
   (A..Street, A..Number, A..City, A..State)

CREATE FUNCTION US_addresstofunc(A US_addr_t)
```

```
   RETURNS ROW
   (Street VARCHAR(30), Number CHAR(15), City
   VARCHAR(30), STATE VARCHAR (10), Zip
   CHAR(10))
   LANGUAGE SQL
   RETURN VALUES
   (A..Street, A..Number, A..City, A..State, A..Zip)
```

Paso 2. Cree grupos de transformación, uno para cada variación de tipo:

```
CREATE TRANSFORM FOR Address_t
   funcgroup1 (FROM SQL WITH FUNCTION addresstofunc)

CREATE TRANSFORM FOR US_addr_t
   funcgroup2 (FROM SQL WITH FUNCTION US_addresstofunc)
```

Paso 3. Cree UDF externas, una para cada variación de tipo.

Registre la UDF externa para el tipo Address_t:

```
CREATE FUNCTION address_to_client (A Address_t)
   RETURNS VARCHAR(150)
   LANGUAGE C
   EXTERNAL NAME 'addressudf!address_to_client'
  ...
   TRANSFORM GROUP funcgroup1
```

Escriba la UDF address_to_client:

```
void SQL_API_FN address_to_client(
   SQLUDF_VARCHAR  *street,
   SQLUDF_CHAR     *number,
   SQLUDF_VARCHAR  *city,
   SQLUDF_VARCHAR  *state,
   SQLUDF_VARCHAR  *output,

   /* Indicadores de nulo para atributos */
   SQLUDF_NULLIND *street_ind,
   SQLUDF_NULLIND  *number_ind,
   SQLUDF_NULLIND  *city_ind,
   SQLUDF_NULLIND  *state_ind,
   /* Indicador de nulo para la instancia */
   SQLUDF_NULLIND *address_ind,
   /* Indicador de nulo para la salida */
   SQLUDF_NULLIND  *output_ind,
   SQLUDF_TRAIL_ARGS)

{
   sprintf (output, "[address_t] [Street:%s] [number:%s]
   [city:%s] [state:%s]",
   street, number, city, state);
   *output_ind = 0;
}
```

Registre la UDF externa para el tipo US_addr_t:

```
CREATE FUNCTION address_to_client (A US_addr_t)
   RETURNS VARCHAR(150)
   LANGUAGE C
   EXTERNAL NAME 'addressudf!US_addr_to_client'
  ...
   TRANSFORM GROUP funcgroup2
```

Escriba la UDF US_addr_to_client:

```
void SQL_API_FN US_address_to_client(
   SQLUDF_VARCHAR  *street,
   SQLUDF_CHAR     *number,
   SQLUDF_VARCHAR  *city,
   SQLUDF_VARCHAR  *state,
   SQLUDF_CHAR     *zip,
   SQLUDF_VARCHAR  *output,
```

```
    /* Indicadores de nulo */
    SQLUDF_NULLIND *street_ind,
    SQLUDF_NULLIND  *number_ind,
    SQLUDF_NULLIND  *city_ind,
    SQLUDF_NULLIND  *state_ind,
    SQLUDF_NULLIND  *zip_ind,
    SQLUDF_NULLIND  *us_address_ind,
    SQLUDF_NULLIND  *output_ind,
    SQLUDF_TRAIL_ARGS)

{
    sprintf (output, "[US_addr_t] [Street:%s] [number:%s]
    [city:%s] [state:%s] [zip:%s]",
    street, number, city, state, zip);
    *output_ind = 0;
}
```

Paso 4. Cree una UDF incorporada al SQL que elija la UDF externa correcta para procesar la instancia. La UDF siguiente utiliza la especificación TREAT en sentencias SELECT combinadas mediante una cláusula UNION ALL para invocar la transformación de cliente FROM SQL correcta:

```
CREATE FUNCTION addr_stream (ab Address_t)
   RETURNS VARCHAR(150)
   LANGUAGE SQL
   RETURN
   WITH temp(addr) AS
   (SELECT address_to_client(ta.a)
      FROM TABLE (VALUES (ab)) AS ta(a)
      WHERE ta.a IS OF (ONLY Address_t)
      UNION ALL
   SELECT address_to_client(TREAT (tb.a AS US_addr_t))
      FROM TABLE (VALUES (ab)) AS tb(a)
      WHERE tb.a IS OF (ONLY US_addr_t))
   SELECT addr FROM temp;
```

En este punto, las aplicaciones pueden invocar la UDF externa apropiada invocando la función `Addr_stream`:

```
SELECT Addr_stream(Address)
   FROM Employee;
```

Paso 5. Añada la UDF externa `Addr_stream` como transformación de *cliente* FROM SQL para `Address_t`:

```
CREATE TRANSFORM GROUP FOR Address_t
   client_group (FROM SQL
   WITH FUNCTION Addr_stream)
```

Nota: Si la aplicación puede utilizar un tipo de predicado para especificar en la consulta tipos de direcciones concretos, añada `Addr_stream` como transformación de cliente FROM SQL para `US_addr_t`. Así se asegurará de que se pueda invocar `Addr_stream` cuando una consulta solicite específicamente instancias de `US_addr_t`.

Paso 6. Enlace lógicamente la aplicación con la opción TRANSFORM GROUP establecida en `client_group`.

```
PREP myprogram TRANSFORM GROUP client_group
```

Cuando DB2 enlaza lógicamente la aplicación que contiene la sentencia `SELECT Address FROM Person INTO :hvar`, DB2 busca una transformación de cliente FROM SQL. DB2 reconoce que se está desenlazando lógicamente un tipo estructurado y busca en el grupo de transformación `client_group`, puesto que éste es el grupo de transformación (TRANSFORM GROUP) especificado durante el enlace lógico en el paso 6.

El grupo de transformación contiene la función de transformación Addr_stream asociada al tipo raíz Address_t en el paso 5 en la página 324. Addr_stream es una función incorporada al SQL, definida en el paso 4 en la página 324, por lo que no depende de ninguna otra función de transformación. La función Addr_stream devuelve VARCHAR(150), el tipo de datos que requiere la variable del lenguaje principal :hvaddr.

La función Addr_stream toma un valor de entrada del tipo Address_t, que en este ejemplo se puede sustituir por US_addr_t, y determina el tipo dinámico del valor de entrada. Cuando Addr_stream determina el tipo dinámico, invoca la UDF externa correspondiente sobre el valor: address_to_client si el tipo dinámico es Address_t; o USaddr_to_client si el tipo dinámico es US_addr_t. Estas dos UDF se definen en el paso 3 en la página 323. Cada UDF descompone su tipo estructurado respectivo en VARCHAR(150), el tipo requerido por la función de transformación Addr_stream.

Para aceptar los tipos estructurados como entrada, cada UDF necesita una función de transformación FROM SQL para descomponer la instancia del tipo estructurado de entrada en parámetros de atributo individuales. Las sentencias CREATE FUNCTION del paso 3 en la página 323 denominan el TRANSFORM GROUP que contiene estas transformaciones.

Las sentencias CREATE FUNCTION para las funciones de transformación se emiten en el paso 1 en la página 322. Las sentencias CREATE TRANSFORM que asocian las funciones de transformación a sus grupos de transformación se emiten en el 2 en la página 323.

Conceptos relacionados:

- "Requisitos de las funciones de transformación" en la página 321
- "Funciones de transformación y grupos de transformación" en la página 306

Información relacionada:

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

Devolución de datos de subtipo a DB2

Suponga que desea insertar un tipo estructurado en una base de datos DB2 desde una aplicación utilizando la sintaxis siguiente:

```
INSERT INTO person (Oid, Name, Address)
   VALUES ('n', 'Norm', :hvaddr);
```

Para ejecutar la sentencia INSERT para un tipo estructurado:

Paso 1. Cree una transformación de función TO SQL para cada variación de la dirección. El ejemplo siguiente muestra las UDF incorporadas al SQL que transforman los tipos Address_t y US_addr_t:

```
CREATE FUNCTION functoaddress
 (str VARCHAR(30), num CHAR(15), cy VARCHAR(30), st VARCHAR (10))
RETURNS Address_t
LANGUAGE SQL
RETURN Address_t()..street(str)..number(num)..city(cy)..state(st);

CREATE FUNCTION functoaddress
 (str VARCHAR(30), num CHAR(15), cy VARCHAR(30), st VARCHAR (10),
  zp CHAR(10))
```

```
RETURNS US_addr_t
LANGUAGE SQL
RETURN US_addr_t()..street(str)..number(num)..city(cy)
    ..state(st)..zip(zp);
```

Paso 2. Cree grupos de transformación, uno para cada variación de tipo:

```
CREATE TRANSFORM FOR Address_t
   funcgroup1 (TO SQL
   WITH FUNCTION functoaddress);

CREATE TRANSFORM FOR US_addr_t
   funcgroup2 (TO SQL
   WITH FUNCTION functousaddr);
```

Paso 3. Cree UDF externas que devuelvan tipos de dirección codificados, una para cada variación de tipo.

Registre la UDF externa para el tipo `Address_t`:

```
CREATE FUNCTION client_to_address (encoding VARCHAR(150))
   RETURNS Address_t
   LANGUAGE C
   TRANSFORM GROUP funcgroup1
   ...
   EXTERNAL NAME 'address!client_to_address';
```

Escriba la UDF externa para la versión `Address_t` de `client_to_address`:

```
void SQL_API_FN client_to_address (
   SQLUDF_VARCHAR *encoding,
   SQLUDF_VARCHAR  *street,
   SQLUDF_CHAR     *number,
   SQLUDF_VARCHAR  *city,
   SQLUDF_VARCHAR  *state,

   /* Indicadores de nulo */
   SQLUDF_NULLIND *encoding_ind,
   SQLUDF_NULLIND *street_ind,
   SQLUDF_NULLIND  *number_ind,
   SQLUDF_NULLIND  *city_ind,
   SQLUDF_NULLIND  *state_ind,
   SQLUDF_NULLIND *address_ind,
   SQLUDF_TRAIL_ARGS )
{
   char c[150];
   char *pc;

   strcpy(c, encoding);

   pc = strtok (c, ":]");
   pc = strtok (NULL, ":]");
   pc = strtok (NULL, ":]");
   strcpy (street, pc);
   pc = strtok (NULL, ":]");
   pc = strtok (NULL, ":]");
   strcpy (number, pc);
   pc = strtok (NULL, ":]");
   pc = strtok (NULL, ":]");
   strcpy (city, pc);
   pc = strtok (NULL, ":]");
   pc = strtok (NULL, ":]");
   strcpy (state, pc);

   *street_ind = *number_ind = *city_ind
   = *state_ind = *address_ind = 0;
}
```

Registre la UDF externa para el tipo `US_addr_t`:

```
CREATE FUNCTION client_to_us_address (encoding VARCHAR(150))
   RETURNS US_addr_t
   LANGUAGE C
   TRANSFORM GROUP funcgroup1
   ...
   EXTERNAL NAME 'address!client_to_US_addr';
```

Escriba la UDF externa para la versión US_addr_t de client_to_address:

```
void SQL_API_FN client_to_US_addr(
   SQLUDF_VARCHAR *encoding,
   SQLUDF_VARCHAR  *street,
   SQLUDF_CHAR     *number,
   SQLUDF_VARCHAR  *city,
   SQLUDF_VARCHAR  *state,
   SQLUDF_VARCHAR *zip,

   /* Indicadores de nulo */
   SQLUDF_NULLIND *encoding_ind,
   SQLUDF_NULLIND *street_ind,
   SQLUDF_NULLIND  *number_ind,
   SQLUDF_NULLIND  *city_ind,
   SQLUDF_NULLIND  *state_ind,
   SQLUDF_NULLIND  *zip_ind,
   SQLUDF_NULLIND *us_addr_ind,
   SQLUDF_TRAIL_ARGS)

{
   char c[150];
   char *pc;

   strcpy(c, encoding);

   pc = strtok (c, ":]");
   pc = strtok (NULL, ":]");
   pc = strtok (NULL, ":]");
   strcpy (street, pc);
   pc = strtok (NULL, ":]");
   pc = strtok (NULL, ":]");
   strncpy (number, pc,14);
   pc = strtok (NULL, ":]");
   pc = strtok (NULL, ":]");
   strcpy (city, pc);
   pc = strtok (NULL, ":]");
   pc = strtok (NULL, ":]");
   strcpy (state, pc);
   pc = strtok (NULL, ":]");
   pc = strtok (NULL, ":]");
   strncpy (zip, pc, 9);

   *street_ind = *number_ind = *city_ind
   = *state_ind = *zip_ind =  *us_addr_ind = 0;
}
```

Paso 4. Cree una UDF incorporada al SQL que elija la UDF externa correcta para procesar esta instancia. La UDF siguiente utiliza el predicado TYPE para invocar la transformación de cliente correcta. El resultado se sitúa en una tabla temporal:

```
CREATE FUNCTION stream_address (ENCODING VARCHAR(150))
   RETURNS Address_t
   LANGUAGE SQL
   RETURN
   (CASE(SUBSTR(ENCODING,2,POSSTR(ENCODING,']')-2))
   WHEN 'address_t'
      THEN client_to_address(ENCODING)
```

```
WHEN 'us_addr_t'
   THEN client_to_us_addr(ENCODING)
ELSE NULL
END);
```

Paso 5. Añada la UDF `stream_address` como transformación de cliente TO SQL para `Address_t`:

```
CREATE TRANSFORM FOR Address_t
   client_group (TO SQL
   WITH FUNCTION stream_address);
```

Paso 6. Enlace lógicamente la aplicación con la opción TRANSFORM GROUP establecida en `client_group`.

```
PREP myProgram2 TRANSFORM GROUP client_group
```

Cuando se enlaza lógicamente la aplicación que contiene la sentencia INSERT con un tipo estructurado, DB2 busca una transformación de cliente TO SQL. DB2 busca la transformación en el grupo de transformación `client_group`, puesto que éste es el TRANSFORM GROUP especificado durante el enlace lógico en el paso 6. DB2 encuentra la función de transformación que necesita: `stream_address`, que se asocia al tipo raíz `Address_t` en el paso 5.

`stream_address` es una función incorporada al SQL, definida en el paso 4 en la página 327, por lo que no tiene establecida ninguna dependencia de ninguna función de transformación adicional. Para los parámetros de entrada, `stream_address` acepta VARCHAR(150), que corresponde a la variable del lenguaje principal `:hvaddr` de la aplicación. `stream_address` devuelve un valor que es tanto del tipo raíz correcto, `Address_t`, como del tipo dinámico correcto.

`stream_address` analiza el parámetro de entrada VARCHAR(150) en busca de una serie que mencione el tipo dinámico: en este caso, pueden ser 'Address_t' o 'US_addr_t'. A continuación, `stream_address` invoca la UDF externa correspondiente para analizar el VARCHAR(150) y devuelve un objeto del tipo especificado. Existen dos UDF `client_to_address()`, una para devolver cada tipo posible. Estas UDF se definen en el paso 3 en la página 326. Cada UDF toma el VARCHAR(150) de entrada e, internamente, construye los atributos del tipo estructurado apropiado, devolviendo así el tipo estructurado.

Para devolver el tipo estructurado, cada UDF necesita una función de transformación TO SQL para construir los valores de los atributos de salida en una instancia del tipo estructurado. Las sentencias CREATE FUNCTION del paso 3 en la página 326 mencionan el TRANSFORM GROUP que contiene las transformaciones.

Las funciones de transformación incorporadas al SQL del paso 1 en la página 325, y las asociaciones con los grupos de transformación del paso 2 en la página 326, se mencionan en las sentencias CREATE FUNCTION del paso 3 en la página 326.

Conceptos relacionados:

- "Requisitos de las funciones de transformación" en la página 321
- "Funciones de transformación y grupos de transformación" en la página 306

Información relacionada:

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

Variables del lenguaje principal de tipo estructurado

Declaración de variables del lenguaje principal de tipo estructurado

Para recuperar o enviar variables del lenguaje principal de tipo estructurado en SQL estático, tiene que proporcionar una declaración SQL que indique el tipo incorporado utilizado para representar el tipo estructurado. El formato de la declaración es el siguiente:

```
EXEC SQL BEGIN DECLARE SECTION ;

   SQL TYPE IS structured_type AS base_type host-variable-name ;

EXEC SQL END DECLARE SECTION;
```

Por ejemplo, suponga que el tipo `Address_t` se va a transformar en un tipo de caracteres de longitud variable cuando se pase a la aplicación cliente. Utilice la declaración siguiente para la variable del lenguaje principal de tipo `Address_t`:

```
SQL TYPE IS Address_t AS VARCHAR(150) addrhv;
```

Conceptos relacionados:

- "Funciones de transformación y grupos de transformación" en la página 306

Tareas relacionadas:

- "Descripción de un tipo estructurado" en la página 329

Descripción de un tipo estructurado

Una DESCRIBE de una sentencia con una variable de tipo estructurado hace que DB2 sitúe una descripción del tipo resultado de la función de transformación FROM SQL en el campo SQLTYPE de la SQLVAR base de la SQLDA. Sin embargo, si no se ha definido ninguna función de transformación FROM SQL, ya sea porque no se ha especificado ningún TRANSFORM GROUP utilizando el registro especial CURRENT DEFAULT TRANSFORM GROUP o porque el grupo mencionado no tiene definida una función de transformación FROM SQL, DESCRIBE devuelve un error.

El nombre real del tipo estructurado se devuelve en SQLVAR2.

Conceptos relacionados:

- "Funciones de transformación y grupos de transformación" en la página 306

Tareas relacionadas:

- "Declaración de variables del lenguaje principal de tipo estructurado" en la página 329

Capítulo 9. Activadores

Activadores en el desarrollo de aplicaciones . . . 331
Activadores INSERT, UPDATE y DELETE 335
Interacciones de los activadores con las restricciones referenciales 336
Activadores INSTEAD OF 336
Directrices para la creación de activadores. . . . 337
Creación de activadores 338
Granularidad del activador 339
Tiempo de activación del activador 340
Variables de transición 343
Tablas de transición 344
Acción activada 346
  Acción activada 346
  Acciones activadas calificadas por condiciones 346
  Acción activada compuesta por sentencias de SQL 347
  Acción activada que contiene una referencia de función o procedimiento. 348
Activadores múltiples 350
Sinergia entre activadores, restricciones y rutinas 351
  Extracción de información de los UDT, las UDF y los LOB con activadores 351
  Prevención de operaciones sobre tablas mediante activadores 352
  Definición de reglas empresariales mediante activadores 353
  Definición de acciones mediante activadores . . 353

Activadores en el desarrollo de aplicaciones

Para poder cambiar el gestor de bases de datos de forma que pase de ser un sistema pasivo a ser un sistema activo, utilice las posibilidades incorporadas en un activador de SQL.

Un *activador* de SQL es un regla con nombre que está asociada con una sola tabla base. Un activador especifica acciones que se deben activar condicionalmente al producirse un suceso activador, el cual consiste en una modificación (INSERT, UPDATE o DELETE) de una tabla dirigida a una tabla base en particular. Además, un activador especifica cuándo la acción activadora va a tener lugar, si antes o después de que se produzca el suceso activador. Los activadores se crean y eliminan con las sentencias CREATE TRIGGER y DROP TRIGGER. La figura siguiente ilustra la estructura lógica básica de un activador:

Se puede considerar que un activador sigue esta lógica: cuando se produzca un suceso, si una condición prescrita es verdadera, ejecuta una acción. El suceso es una operación de base de datos sobre una tabla. La condición puede ser una condición del estado de la base de datos o un estado transicional de la tabla después del suceso de la operación. La acción puede ser la ejecución de una o más sentencias de SQL que efectúen más cambios en la base de datos, la emisión de una excepción para impedir que tenga lugar la operación de modificación, un arreglo de datos modificados en la operación del suceso o cualquier actividad que se pueda incluir lógicamente en una invocación de procedimiento o función. Los procedimientos y funciones pueden contener lógica compleja y se pueden utilizar como subrutinas en el activador. Las funciones externas definidas por el usuario y los procedimientos pueden permitir que un activador envíe un mensaje de correo electrónico o que grabe datos en un archivo del sistema de archivos.

El diagrama siguiente muestra la estructura lógica de un activador:

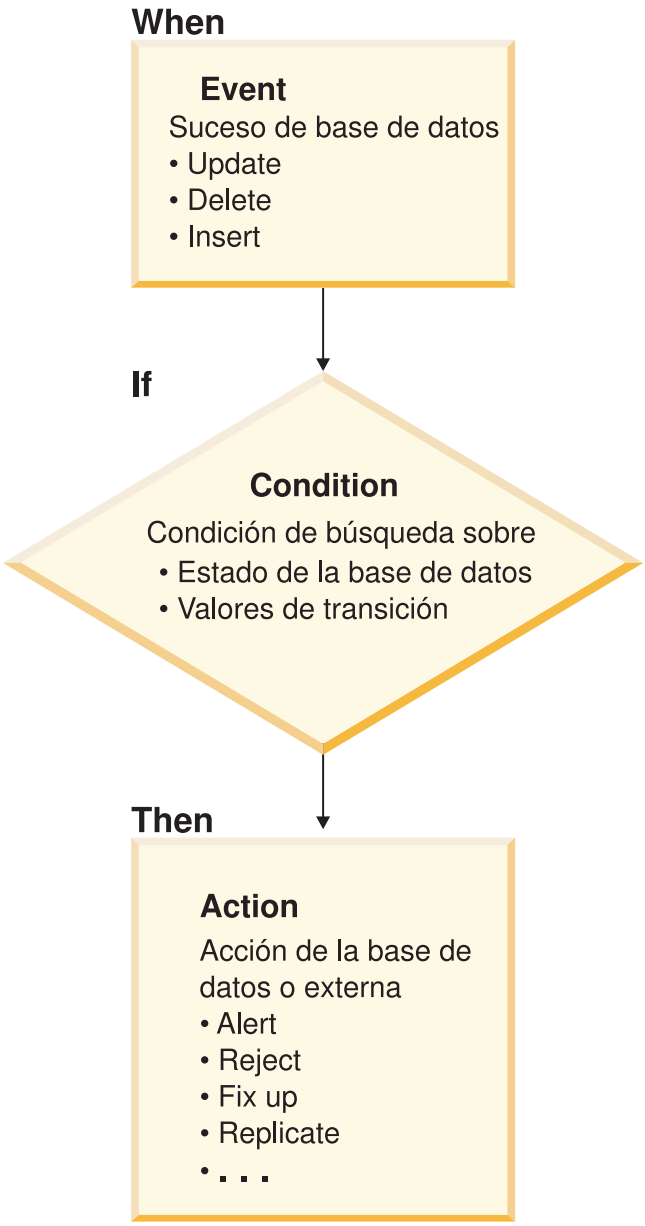


Figura 16. Clasificaciones de las rutinas

Puede utilizar activadores para dar soporte a formas generales de integridad, como por ejemplo las reglas empresariales. Por ejemplo, puede que la empresa desee rechazar pedidos que sobrepasen el límite de crédito de sus clientes. Se puede utilizar un activador para imponer esta restricción. En general, los activadores son potentes mecanismos para capturar reglas empresariales *transicionales*. Las reglas empresariales transicionales son reglas que implican distintos estados de los datos.

Por ejemplo, suponga que un salario no se puede incrementar en más del 10 por ciento. Para comprobar esta regla, se debe comparar el valor del salario antes y después del incremento. En el caso de las reglas que no implican más de un estado de los datos, resultan más apropiadas las restricciones de comprobación e integridad referencial. Debido a la semántica declarativa de las restricciones referenciales y de comprobación, se recomienda su uso para las restricciones que no son transicionales.

También puede utilizar activadores para tareas tales como una actualización automática de los datos de resumen. Manteniendo estas acciones formando parte de la base de datos y asegurándose de que se producen automáticamente, los activadores mejoran la integridad de la base de datos. Por ejemplo, suponga que desea hacer un seguimiento automático del número de empleados gestionados por una empresa:

```
Tablas: EMPLOYEE (de las tablas Sample)
  COMPANY_STATS (NBEMP, NBPRODUCT, REVENUE)
```

Puede definir dos activadores:

- Un activador que incremente el número de empleados cada vez que se contrate a una nueva persona; es decir, cada vez que se inserte una fila en la tabla EMPLOYEE:

  ```
  CREATE TRIGGER NEW_HIRED
    AFTER INSERT ON EMPLOYEE
    FOR EACH ROW MODE DB2SQL
    UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1
  ```

- Un activador que decremente el número de empleados cada vez que un empleado deje la empresa; es decir, cada vez que se suprima una fila de la tabla EMPLOYEE:

  ```
  CREATE TRIGGER FORMER_EMP
    AFTER DELETE ON EMPLOYEE
    FOR EACH ROW MODE DB2SQL
    UPDATE COMPANY_STATS SET NBEMP = NBEMP - 1
  ```

Específicamente, puede utilizar activadores para:

- Validar datos de entrada utilizando la sentencia SIGNAL SQLSTATE de SQL y la función incorporada RAISE_ERROR o invocar un procedimiento almacenado (sólo de serie) o UDF para devolver un SQLSTATE que indique que se ha producido un error si se descubren datos no válidos. Observe que una validación de datos no transicionales se suele manejar mejor mediante restricciones referenciales y de comprobación. En cambio, los activadores resultan apropiados para validar datos transicionales; es decir, para las validaciones que requieren comparaciones entre los valores anterior y posterior a una operación de actualización.
- Generar automáticamente valores para filas recién insertadas (esto se conoce como una *función suplente*). Es decir, para implementar valores por omisión definidos por el usuario, posiblemente en base a otros valores de la fila o de otras tablas. Para implementar columnas dependientes funcionalmente, DB2® también da soporte a columnas generadas (GENERATED). Se trata de columnas cuyos valores siempre se deducen de forma determinista a partir de otros valores de la misma fila.
- Leer otras tablas a efectos de realizar referencias cruzadas.
- Grabar otras tablas a efectos de realizar seguimientos de comprobación.
- Soportar *alertas* (por ejemplo, mediante mensajes de correo electrónico).

La utilización de activadores en el gestor de bases de datos puede tener como consecuencia:

- **Creación rápida de aplicaciones**.

  Puesto que los activadores se almacenan en la base de datos relacional, las acciones realizadas por éstos no se tienen que codificar en cada una de las aplicaciones.
- **Aplicación global de las reglas empresariales**

Un activador sólo se tiene que definir una vez, y luego lo puede utilizar cualquier aplicación que cambie la tabla.

- **Facilidad de mantenimiento**

  Si cambia la política de una empresa, sólo es necesario cambiar el activador correspondiente, en lugar de tener que cambiar cada uno de los programas de aplicación.

Cuando se ejecuta una sentencia de SQL activada, ésta puede ocasionar que se produzca otro suceso activador, o incluso el mismo, el cual, a su vez, hace que se active el otro activador (o una segunda instancia del mismo). Por consiguiente, la activación de un activador puede activar en cascada otro u otros activadores.

El número de niveles de activadores en cascada soportado es de 16. Si se activa un activador al nivel 17, se devolverá SQLCODE -724 (SQLSTATE 54038) y se retrotraerá la sentencia activadora.

Conceptos relacionados:

- "Activadores INSERT, UPDATE y DELETE" en la página 335
- "Granularidad del activador" en la página 339
- "Tiempo de activación del activador" en la página 340
- "Interacciones de los activadores con las restricciones referenciales" en la página 336
- "Directrices para la creación de activadores" en la página 337
- "Activadores INSTEAD OF" en la página 336

Tareas relacionadas:

- "Creación de activadores" en la página 338
- "Definición de reglas empresariales mediante activadores" en la página 353

Información relacionada:

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

Ejemplos relacionados:

- "tbtrig.out -- HOW TO USE TRIGGERS (C)"
- "tbtrig.sqc -- How to use a trigger on a table (C)"
- "tbtrig.out -- HOW TO USE TRIGGERS (C++)"
- "tbtrig.sqC -- How to use a trigger on a table (C++)"
- "trigsql.sqb -- How to use a trigger on a table (IBM COBOL)"
- "TbTrig.java -- How to use triggers (JDBC)"
- "TbTrig.out -- HOW TO USE TRIGGERS. Connect to 'sample' database using JDBC type 2 driver (JDBC)"
- "TbTrig.out -- HOW TO USE TRIGGERS. Connect to 'sample' database using JDBC type 2 driver (SQLJ)"
- "TbTrig.sqlj -- How to use triggers (SQLj)"

Activadores INSERT, UPDATE y DELETE

Cada activador está asociado a un suceso. Los activadores se activan cuando se produce el suceso correspondiente en la base de datos. Este suceso activador sucede cuando se realiza la acción especificada, ya sea UPDATE, INSERT o DELETE (incluyendo las ocasionadas por acciones de restricciones referenciales) sobre la tabla sujeto. Por ejemplo:

```
CREATE TRIGGER NEW_HIRE
      AFTER INSERT ON EMPLOYEE
    FOR EACH ROW
      UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1
```

La sentencia anterior define el activador `new_hire`, el cual se activa cuando se realiza una operación de inserción sobre la tabla `employee`.

Se asocia cada suceso activador, y consecuentemente cada activador, con exactamente una tabla sujeto y exactamente una operación de modificación. Las operaciones de modificación son:

Operación de inserción
Una operación de inserción sólo puede ser ocasionada por una sentencia INSERT. Por consiguiente, no se activan los activadores cuando se cargan datos utilizando programas de utilidad que no utilizan INSERT, como por ejemplo el mandato LOAD.

Operación de actualización
Una operación de actualización puede ser ocasionada por una sentencia UPDATE o como consecuencia de una regla de restricción referencial de ON DELETE SET NULL.

Operación de supresión
Una operación de actualización puede ser ocasionada por una sentencia DELETE o como consecuencia de una regla de restricción referencial de ON DELETE CASCADE.

Si el suceso activador es una operación de actualización, el suceso se puede asociar a columnas específicas de la tabla sujeto. En este caso, el activador sólo se activa si la operación de actualización intenta actualizar cualquiera de las columnas especificadas. Esto brinda un refinamiento adicional del suceso que activa el activador.

Por ejemplo, el activador siguiente, `REORDER` , sólo se activa si se realiza una operación de actualización sobre las columnas `ON_HAND` o `MAX_STOCKED`, de la tabla `PARTS`.

```
CREATE TRIGGER REORDER
  AFTER UPDATE OF ON_HAND, MAX_STOCKED ON PARTS
  REFERENCING NEW AS N_ROW
FOR EACH ROW
  WHEN (N_ROW.ON_HAND < 0.10 * N_ROW.MAX_STOCKED)
  BEGIN ATOMIC
  VALUES(ISSUE_SHIP_REQUEST(N_ROW.MAX_STOCKED -
                            N_ROW.ON_HAND,
                            N_ROW.PARTNO));
  END
```

Conceptos relacionados:

- "Granularidad del activador" en la página 339
- "Tiempo de activación del activador" en la página 340

- "Interacciones de los activadores con las restricciones referenciales" en la página 336
- "Activadores en el desarrollo de aplicaciones" en la página 331
- "Activadores INSTEAD OF" en la página 336

Tareas relacionadas:

- "Creación de activadores" en la página 338

Interacciones de los activadores con las restricciones referenciales

Un suceso activador puede ser consecuencia de cambios debidos a la imposición de restricciones referenciales. Por ejemplo, supongamos dos tablas, DEPT y EMP; si una supresión o actualización de DEPT ocasiona supresiones o actualizaciones propagadas en EMP por medio de restricciones de integridad referenciales, los activadores de supresión o actualización definidos sobre EMP se activan como consecuencia de la restricción referencial definida sobre DEPT. Los activadores sobre EMP se ejecutar BEFORE (antes) o AFTER (después) de la supresión (en el caso de ON DELETE CASCADE) o de la actualización de filas en EMP (en el caso de ON DELETE SET NULL), en función de su tiempo de activación correspondiente.

Conceptos relacionados:

- "Activadores INSERT, UPDATE y DELETE" en la página 335
- "Granularidad del activador" en la página 339
- "Activadores en el desarrollo de aplicaciones" en la página 331

Activadores INSTEAD OF

Los activadores INSTEAD OF describen cómo realizar operaciones de inserción, actualización y supresión sobre vistas que son demasiado complejas para soportar estas operaciones de forma nativa. Los activadores INSTEAD OF permiten que las aplicaciones utilicen una vista como interfaz exclusiva para todas las operaciones de SQL (inserción, supresión, actualización y selección). Normalmente, los activadores INSTEAD OF contienen la antítesis de la lógica aplicada en el cuerpo de una vista. Por ejemplo, supongamos una vista que descifra columnas de su tabla fuente. El activador INSTEAD OF para esta vista cifra datos y luego los inserta en la tabla fuente, realizando así la operación simétrica.

Mediante el uso de un activador INSTEAD OF, la operación de modificación solicitada sobre la vía se sustituye por la lógica del activador, que realiza la operación en nombre de la vista. Desde la perspectiva de la aplicación, esto sucede de forma transparente, ya que percibe que todas las operaciones se realizan sobre la vista. Sólo se permite un activador INSTEAD OF para cada tipo de operación sobre una vista sujeto determinada.

La propia vista tiene que ser una vista sin tipo o un alias que se resuelva en una vista sin tipo. Asimismo, no puede ser una vista que se defina utilizando WITH CHECK OPTION (vista simétrica) ni una vista sobre la que se haya definido, directa o indirectamente, una vista simétrica.

El ejemplo siguiente presenta tres activadores INSTEAD OF que proporcionan lógica para inserciones (INSERT), actualizaciones (UPDATE) y supresiones

(DELETE) para la vista definida (EMPV). La vista EMPV contiene una unión en su cláusula FROM y, por consiguiente, no puede soportar ninguna operación de modificación de forma nativa.

```
CREATE VIEW EMPV(EMPNO, FIRSTNME, MIDINIT, LASTNAME, PHONENO,
                 HIREDATE, DEPTNAME)
AS SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, PHONENO,
          HIREDATE, DEPTNAME
          FROM EMPLOYEE, DEPARTMENT WHERE
               EMPLOYEE.WORKDEPT = DEPARTMENT.DEPTNO

CREATE TRIGGER EMPV_INSERT INSTEAD OF INSERT ON EMPV
REFERENCING NEW AS NEWEMP FOR EACH ROW
INSERT INTO EMPLOYEE (EMPNO, FIRSTNME, MIDINIT, LASTNAME,
                      WORKDEPT, PHONENO, HIREDATE)
        VALUES(EMPNO, FIRSTNME, MIDINIT, LASTNAME,
               COALESCE((SELECT DEPTNO FROM DEPARTMENT AS D
                         WHERE D.DEPTNAME = NEWEMP.DEPTNAME),
                        RAISE_ERROR('70001', 'Unknown dept name')),
               PHONENO, HIREDATE)

CREATE TRIGGER EMPV_UPDATE INSTEAD OF UPDATE ON EMPV
REFERENCING NEW AS NEWEMP OLD AS OLDEMP
    FOR EACH ROW
BEGIN ATOMIC
 VALUES(CASE WHEN NEWEMP.EMPNO = OLDEMP.EMPNO THEN 0
             ELSE RAISE_ERROR('70002', 'Must not change EMPNO') END);
 UPDATE EMPLOYEE AS E
   SET (FIRSTNME, MIDINIT, LASTNAME, WORKDEPT, PHONENO, HIREDATE)
     = (NEWEMP.FIRSTNME, NEWEMP.MIDINIT, NEWEMP.LASTNAME,
               COALESCE((SELECT DEPTNO FROM DEPARTMENT AS D
                         WHERE D.DEPTNAME = NEWEMP.DEPTNAME),
                        RAISE_ERROR ('70001', 'Unknown dept name')),
               NEWEMP.PHONENO, NEWEMP.HIREDATE)
 WHERE NEWEMP.EMPNO = E.EMPNO;
END

CREATE TRIGGER EMPV_DELETE INSTEAD OF DELETE ON EMPV
REFERENCING OLD AS OLDEMP FOR EACH ROW
DELETE FROM EMPLOYEE AS E WHERE E.EMPNO = OLDEMP.EMPNO
```

Conceptos relacionados:

- "Activadores INSERT, UPDATE y DELETE" en la página 335
- "Activadores en el desarrollo de aplicaciones" en la página 331

Tareas relacionadas:

- "Creación de activadores" en la página 338

Información relacionada:

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

Directrices para la creación de activadores

Cuando se crea un activador, se le debe asociar a una tabla. Esta tabla recibe el nombre de *tabla sujeto* del activador. El término *operación de modificación* hace referencia a cualquier cambio en el estado de la tabla sujeto. Inician una operación de modificación::

- una sentencia INSERT.
- una sentencia UPDATE o una restricción referencial que ejecuta una UPDATE (actualización)

- una sentencia DELETE o una restricción referencial que ejecuta una DELETE (supresión)

Debe asociar cada activador a uno de estos tres tipos de operaciones de modificación. La asociación se denomina *suceso activador* para ese activador en concreto.

También debe definir la acción, denominada *acción activada*, que el activador realiza cuando se produce el suceso activador correspondiente. La acción activada consta de una o más sentencias de SQL que se pueden ejecutar antes o después de que el gestor de bases de datos realice el suceso activador. Una vez que se produce un suceso activador, el gestor de bases de datos determina el conjunto de filas de la tabla sujeto a las que afecta la operación de modificación y ejecuta el activador.

Cuando cree un activador, debe declarar los atributos y el comportamiento siguientes:

- El nombre del activador.
- El nombre de la tabla sujeto.
- El tiempo de activación del activador (BEFORE (antes) o AFTER (después) de que se ejecute la operación de modificación).
- El suceso activador (INSERT (inserción), DELETE (supresión) o UPDATE (actualización)).
- La variable de transición de los valores antiguos, si la hay.
- La variable de transición de los valores nuevos, si la hay.
- La tabla de transición de los valores antiguos, si la hay.
- La tabla de transición de los valores nuevos, si la hay.
- La granularidad (FOR EACH STATEMENT (para cada sentencia) o FOR EACH ROW (para cada fila)).
- La acción activada por el activador (incluyendo una condición de acción activada y una o más sentencias de SQL activada(s)).
- Si el suceso activador es una UPDATE, la lista de columnas activadoras para el suceso activador, así como una indicación de si la lista de columnas activadoras era explícita o implícita.

Conceptos relacionados:

- "Activadores INSERT, UPDATE y DELETE" en la página 335
- "Granularidad del activador" en la página 339
- "Tiempo de activación del activador" en la página 340
- "Activadores en el desarrollo de aplicaciones" en la página 331

Tareas relacionadas:

- "Creación de activadores" en la página 338

Información relacionada:

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

Creación de activadores

Para crear un activador desde el Centro de control, utilice el diálogo Crear activador. Puede encontrar el diálogo Crear activador expandiendo el árbol de objetos y pulsando la carpeta Activadores con el botón derecho del ratón.

Para crear un activador utilizando la línea de mandatos, utilice la plantilla siguiente de la sentencia CREATE TRIGGER:

```
CREATE TRIGGER <nombre>
 <acción> ON <nombre_tabla>
 <operación>
 <acción_activada>
```

La sentencia de SQL siguiente crea un activador que incrementa el número de empleados cada vez que se contrata a una nueva persona, sumando 1 a la columna de número de empleados (NBEMP) de la tabla COMPANY_STATS cada vez que se añade una fila a la tabla EMPLOYEE.

```
CREATE TRIGGER NEW_HIRED
 AFTER INSERT ON EMPLOYEE
 FOR EACH ROW
 UPDATE COMPANY_STATS SET NBEMP = NBEMP+1;
```

Conceptos relacionados:

- "Activadores INSERT, UPDATE y DELETE" en la página 335
- "Granularidad del activador" en la página 339
- "Tiempo de activación del activador" en la página 340
- "Activadores en el desarrollo de aplicaciones" en la página 331
- "Directrices para la creación de activadores" en la página 337

Información relacionada:

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

Ejemplos relacionados:

- "tbtrig.out -- HOW TO USE TRIGGERS (C)"
- "tbtrig.sqc -- How to use a trigger on a table (C)"
- "tbtrig.out -- HOW TO USE TRIGGERS (C++)"
- "tbtrig.sqC -- How to use a trigger on a table (C++)"
- "trigsql.sqb -- How to use a trigger on a table (IBM COBOL)"
- "TbTrig.java -- How to use triggers (JDBC)"
- "TbTrig.out -- HOW TO USE TRIGGERS. Connect to 'sample' database using JDBC type 2 driver (JDBC)"
- "TbTrig.out -- HOW TO USE TRIGGERS. Connect to 'sample' database using JDBC type 2 driver (SQLJ)"
- "TbTrig.sqlj -- How to use triggers (SQLj)"

Granularidad del activador

Cuando se activa un activador, éste se ejecuta según su granularidad, de la forma siguiente:

FOR EACH ROW
Se ejecuta tantas veces como número de filas haya en el conjunto de filas afectadas. Si tiene necesidad de hacer referencia a las filas específicas afectadas por la acción activada, utilice la granularidad FOR EACH ROW. Un ejemplo de este caso es la comparación de los valores nuevo y antiguo de una fila actualizada en un activador AFTER UPDATE.

FOR EACH STATEMENT
Se ejecuta una vez para el suceso activador entero.

Si el conjunto de filas afectadas está vacío (es decir, en el caso de una UPDATE o DELETE buscada en la que la cláusula WHERE no califique ninguna fila), un activador FOR EACH ROW no se ejecuta. Pero un activador FOR EACH STATEMENT se sigue ejecutando una vez.

Por ejemplo, se puede mantener la cuenta del número de empleados utilizando FOR EACH ROW.

```
CREATE TRIGGER NEW_HIRED
  AFTER INSERT ON EMPLOYEE
  FOR EACH ROW
  UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1
```

Puede lograr el mismo efecto con una actualización, utilizando la granularidad FOR EACH STATEMENT.

```
CREATE TRIGGER NEW_HIRED
  AFTER INSERT ON EMPLOYEE
  REFERENCING NEW_TABLE AS NEWEMPS
  FOR EACH STATEMENT
  UPDATE COMPANY_STATS
  SET NBEMP = NBEMP + (SELECT COUNT(*) FROM NEWEMPS)
```

Nota: La granularidad FOR EACH STATEMENT no se soporta para los activadores BEFORE.

Conceptos relacionados:

- "Activadores INSERT, UPDATE y DELETE" en la página 335
- "Tiempo de activación del activador" en la página 340
- "Activadores en el desarrollo de aplicaciones" en la página 331
- "Directrices para la creación de activadores" en la página 337

Tareas relacionadas:

- "Creación de activadores" en la página 338

Tiempo de activación del activador

El *tiempo de activación del activador* especifica cuándo se debe activar el activador. Es decir, BEFORE (antes), AFTER (después) o INSTEAD OF (en lugar) de que se ejecute el suceso activador. Por ejemplo, el tiempo de activación del activador siguiente es AFTER la operación INSERT sobre `employee`.

```
CREATE TRIGGER NEW_HIRE
  AFTER INSERT ON EMPLOYEE
FOR EACH ROW
  UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1
```

Si el tiempo de activación es BEFORE, las acciones activadas se activan para cada fila del conjunto de filas afectadas antes de que se ejecute el suceso activador. Por eso, la tabla sujeto sólo se modificará después de que haya finalizado la ejecución del activador BEFORE para cada fila. Observe que los activadores BEFORE deben tener la granularidad FOR EACH ROW.

Si el tiempo de activación es AFTER, las acciones activadas se activan para cada fila del conjunto de filas afectadas o para la sentencia, en función de la granularidad del activador. Esto sucede después de que se ejecute el suceso activador y después de que el gestor de bases de datos compruebe todas las restricciones a las que puede afectar el suceso activador, incluidas las acciones de

restricciones referenciales. Observe que los activadores AFTER pueden tener las granularidades FOR EACH ROW o FOR EACH STATEMENT.

Si el tiempo de activación es INSTEAD OF, las acciones activadas se activan para cada fila del conjunto de filas afectadas en lugar de que se ejecute el suceso activador. Los activadores INSTEAD OF deben tener la granularidad FOR EACH ROW y la tabla sujeto tiene que ser una vista. Ningún otro activador puede utilizar una vista como tabla sujeto.

El diagrama siguiente ilustra el modelo de ejecución de los activadores BEFORE y AFTER:

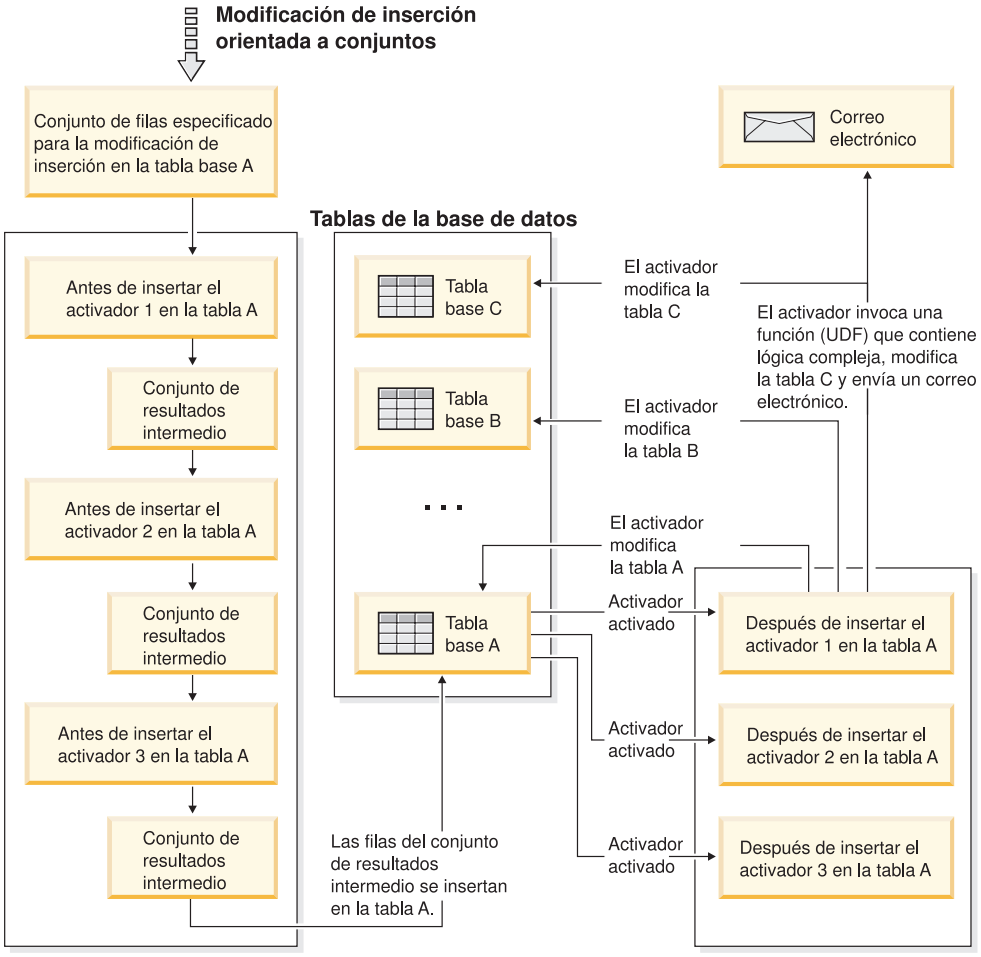


Figura 17. Clasificaciones de las rutinas

Para una tabla determinada con activadores BEFORE y AFTER y un suceso modificador asociado con estos activadores, se activan primero todos los activadores BEFORE. El primer activador BEFORE activado para un suceso determinado opera sobre el conjunto de filas de destino y efectúa en el mismo las modificaciones de actualización prescritas por su lógica. La salida de este activador BEFORE se acepta como entrada en el siguiente activador BEFORE. Cuando se han desencadenado todos los activadores BEFORE activados por el suceso, se aplica a la tabla base el conjunto de resultados intermedio, el resultado de las modificaciones de activador BEFORE en las filas de destino de la operación del suceso activador. Luego, se desencadena cada activador AFTER asociado con el

suceso. Los activadores AFTER pueden modificar la misma u otra tabla o bien pueden realizar una acción externa a la base de datos.

Los distintos tiempos de activación de los activadores reflejan distintos propósitos de los mismos. Básicamente, los activadores BEFORE son una extensión del subsistema de restricciones del sistema de gestión de la base de datos. Por consiguiente, generalmente se utilizan para:
- Realizar una validación de los datos de entrada,
- Generar automáticamente valores para filas recién insertadas
- Leer otras tablas a efectos de realizar referencias cruzadas.

Los activadores BEFORE no se utilizan para modificar adicionalmente la base de datos, puesto que se activan antes de que se aplique a la base de datos el suceso activador. En consecuencia, se activan antes de que se comprueben las restricciones de la integridad.

Y, a la inversa, los activadores AFTER se pueden ver como un módulo de lógica de aplicación que se ejecuta en la base de datos cada vez que se produce un suceso específico. Formando parte de una aplicación, los activadores AFTER siempre ven la base de datos en un estado coherente. Tenga en cuenta que se ejecutan después de las validaciones de las restricciones de la integridad. En consecuencia, los puede utilizar, principalmente, para realizar operaciones que una aplicación también puede efectuar. Por ejemplo:
- Realizar operaciones de modificación consecuentes en la base de datos
- Realizar acciones externas a la base de datos, por ejemplo soportar alertas. Observe que las acciones realizadas fuera de la base de datos no se retrotraen si se retrotrae el activador.

En cambio, puede contemplar un activador INSTEAD OF como una descripción de la operación inversa de la vista sobre la que está definido. Por ejemplo, si la lista de selección de la vista contiene una expresión sobre una tabla base, la sentencia INSERT del cuerpo de su activador INSTEAD OF INSERT contendrá la expresión invertida.

Debido a la naturaleza distinta de los activadores BEFORE, AFTER e INSTEAD OF, se puede utilizar un conjunto diferente de operaciones de SQL para definir las acciones activadas de los activadores BEFORE, AFTER e INSTEAD OF. Por ejemplo, las operaciones de actualización no están permitidas en los activadores BEFORE porque no se garantiza que la acción activada no vaya a violar las restricciones de la integridad. De forma parecida, en los activadores BEFORE, AFTER e INSTEAD OF se soportan distintas granularidades de los activadores. Por ejemplo, FOR EACH STATEMENT no está permitido en los activadores BEFORE porque no se garantiza que la acción activada no violará restricciones, lo cual, a su vez, tendría como resultado una anomalía de la operación.

La sentencia de SQL activada de todos los activadores puede ser una sentencia dinámica compuesta. Sin embargo, los activadores BEFORE se enfrentan con algunas restricciones; no pueden contener las sentencias de SQL siguientes:
- UPDATE
- DELETE
- INSERT

Conceptos relacionados:
- “Activadores INSERT, UPDATE y DELETE” en la página 335

- "Granularidad del activador" en la página 339
- "Acción activada compuesta por sentencias de SQL" en la página 347
- "Activadores en el desarrollo de aplicaciones" en la página 331
- "Directrices para la creación de activadores" en la página 337

Tareas relacionadas:

- "Creación de activadores" en la página 338

Variables de transición

Cuando se implementa un activador FOR EACH ROW, puede que sea necesario hacer referencia al valor de las columnas de la fila en el conjunto de filas afectadas, para las cuales se está ejecutando actualmente el activador. Observe que, para hacer referencia a las columnas de las tablas de la base de datos (incluida la tabla sujeto), puede utilizar sentencias SELECT. Un activador FOR EACH ROW puede hacer referencia a las columnas de la fila para la cual se está ejecutando actualmente utilizando dos variables de transición que se pueden especificar en la cláusula REFERENCING de una sentencia CREATE TRIGGER. Existen dos tipos de variables de transición, que se especifican como *OLD* y *NEW*, junto con un nombre de correlación. Tienen la semántica siguiente:

OLD AS nombre-correlación
: Especifica un nombre de correlación que captura el estado original de la fila, es decir, antes de que se aplique a la base de datos la acción activada.

NEW AS nombre-correlación
: Especifica un nombre de correlación que captura el valor que se utiliza, o se ha utilizado, para actualizar la fila en la base de datos cuando se aplique a la base de datos la acción activada.

Considere el ejemplo siguiente:

```
CREATE TRIGGER REORDER
  AFTER UPDATE OF ON_HAND, MAX_STOCKED ON PARTS
  REFERENCING NEW AS N_ROW
FOR EACH ROW
  WHEN (N_ROW.ON_HAND < 0.10 * N_ROW.MAX_STOCKED
  AND N_ROW.ORDER_PENDING = 'N')
  BEGIN ATOMIC
    VALUES(ISSUE_SHIP_REQUEST(N_ROW.MAX_STOCKED -
                              N_ROW.ON_HAND,
                              N_ROW.PARTNO));
    UPDATE PARTS SET PARTS.ORDER_PENDING = 'Y'
    WHERE PARTS.PARTNO = N_ROW.PARTNO;
  END
```

Basándonos en la definición de las variables de transición OLD y NEW indicadas anteriormente, resulta claro que no se puede definir cada variable de transición para cada activador. Se pueden definir variables de transición en función del tipo de suceso activador:

UPDATE
: Un activador UPDATE puede hacer referencia a variables de transición tanto OLD como NEW.

INSERT
: Un activador INSERT sólo puede hacer referencia a una variable de transición NEW porque, antes de la activación de la operación INSERT, la

fila afectada no existe en la base de datos. Es decir, no existe un estado original de la fila que definiría los valores antiguos antes de que se aplique la acción activada a la base de datos.

DELETE
Un activador DELETE sólo puede hacer referencia a una variable de transición OLD porque en la operación de supresión no se especifican valores nuevos.

Nota: Sólo se pueden especificar variables de transición para los activadores FOR EACH ROW. En un activador FOR EACH STATEMENT, una referencia a una variable de transición no basta para especificar a qué filas del conjunto de filas afectadas hace referencia la variable de transición.

Conceptos relacionados:

- "Activadores INSERT, UPDATE y DELETE" en la página 335
- "Granularidad del activador" en la página 339
- "Tablas de transición" en la página 344

Tareas relacionadas:

- "Creación de activadores" en la página 338

Información relacionada:

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

Tablas de transición

En ambos activadores, FOR EACH ROW y FOR EACH STATEMENT, puede ser necesario hacer referencia al conjunto completo de filas afectadas. Lo es, por ejemplo, si el cuerpo del activador tiene que aplicar agregaciones al conjunto de filas afectadas (por ejemplo, MAX, MIN o AVG de los valores de algunas columnas). Un activador puede hacer referencia al conjunto de filas afectadas utilizando dos tablas de transición que se pueden especificar en la cláusula REFERENCING de una sentencia CREATE TRIGGER. Al igual que en las variables de transición, existen dos tipos de tablas de transición, que se especifican como OLD_TABLE y NEW_TABLE junto con un *nombre-tabla*, con la semántica siguiente:

OLD_TABLE AS nombre-tabla
Especifica el nombre de la tabla que captura el estado original del conjunto de filas afectadas (es decir, antes de que se aplique a la base de datos la operación de SQL activadora).

NEW_TABLE AS nombre-tabla
Especifica el nombre de la tabla que captura el valor que se utiliza para actualizar las filas en la base de datos cuando se aplique a la base de datos la acción activada.

Por ejemplo:

```
CREATE TRIGGER REORDER
  AFTER UPDATE OF ON_HAND, MAX_STOCKED ON PARTS
  REFERENCING NEW_TABLE AS N_TABLE
  NEW AS N_ROW
FOR EACH ROW
  WHEN ((SELECT AVG (ON_HAND) FROM N_TABLE) > 35)
  BEGIN ATOMIC
```

```
        VALUES(INFORM_SUPERVISOR(N_ROW.PARTNO,
                                 N_ROW.MAX_STOCKED,
                                 N_ROW.ON_HAND));
    END
```

Observe que NEW_TABLE siempre contiene el conjunto de filas actualizadas, incluso en un activador FOR EACH ROW. Cuando un activador actúa sobre la tabla sobre la que está definido el activador, NEW_TABLE contiene las filas cambiadas a partir de la sentencia que ha activado el activador. Sin embargo, NEW_TABLE no contiene las filas cambiadas ocasionadas por sentencias internas del activador, puesto que ello ocasionaría una activación separada del activador.

Las tablas de transición sólo son de lectura. A las tablas de transición se aplican las mismas reglas que definen los tipos de variables de transición que se pueden definir para cada suceso activador:

UPDATE
Un activador UPDATE puede hacer referencia a tablas de transición tanto OLD_TABLE como NEW_TABLE.

INSERT
Un activador INSERT sólo puede hacer referencia a una tabla de transición NEW_TABLE porque, antes de la activación de la operación INSERT, las filas afectadas no existen en la base de datos. Es decir, no existe un *estado original de las filas* que defina los valores antiguos antes de que se aplique la acción activada a la base de datos.

DELETE
Un activador DELETE sólo puede hacer referencia a una tabla de transición OLD porque en la operación de supresión no se especifican valores nuevos.

Nota: Es importante observar que se pueden especificar tablas de transición para las dos granularidades de los activadores AFTER: FOR EACH ROW y FOR EACH STATEMENT.

El ámbito de OLD_TABLE y de NEW_TABLE *nombre-tabla* es el cuerpo del activador. En este ámbito, este nombre tiene prioridad sobre el nombre de cualquier otra tabla con el mismo *nombre-tabla* no calificado que pueda existir en el esquema. Por consiguiente, si la OLD_TABLE o NEW_TABLE *nombre-tabla* es, por ejemplo, X, una referencia a X (es decir, una X no calificada) en la cláusula FROM de una sentencia SELECT, siempre hará referencia a la tabla de transición, aunque exista una tabla denominada X en el esquema del creador del activador. En este caso, el usuario debe hacer uso del nombre completamente calificado para hacer referencia a la tabla X del esquema.

Conceptos relacionados:

- "Activadores INSERT, UPDATE y DELETE" en la página 335
- "Granularidad del activador" en la página 339
- "Variables de transición" en la página 343

Tareas relacionadas:

- "Creación de activadores" en la página 338

Información relacionada:

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

Acción activada

Acción activada

Acción activada

La activación de un activador tiene como consecuencia la ejecución de la acción activada asociada al mismo. Cada activador tiene exactamente una acción activada que, a su vez, tiene dos componentes:
- Una *condición de acción activada* especial o una cláusula WHEN
- Un conjunto de *sentencia(s) de SQL activada(s).*

La condición de la acción activada define si se realiza o no se realiza el conjunto de sentencias activadas para la fila o sentencia para las que se está ejecutando la acción activada. El conjunto de sentencias activadas define el conjunto de acciones realizadas por el activador en la base de datos como consecuencia de que se haya producido el suceso correspondiente.

Por ejemplo, la acción activadora siguiente especifica que el conjunto de sentencias de SQL activadas sólo se debe activar para las filas en las que el valor de la columna `on_hand` sea inferior al diez por ciento del valor de la columna `max_stocked`. En este caso, el conjunto de sentencias de SQL activadas es la invocación de la función `issue_ship_request`.

```
CREATE TRIGGER REORDER
  AFTER UPDATE OF ON_HAND, MAX_STOCKED ON PARTS
  REFERENCING NEW AS N_ROW
FOR EACH ROW
        WHEN (N_ROW.ON_HAND < 0.10 * N_ROW.MAX_STOCKED)
  BEGIN ATOMIC
    VALUES(ISSUE_SHIP_REQUEST(N_ROW.MAX_STOCKED -
                              N_ROW.ON_HAND,
                              N_ROW.PARTNO));
  END
```

Conceptos relacionados:

- "Acciones activadas calificadas por condiciones" en la página 346
- "Acción activada compuesta por sentencias de SQL" en la página 347
- "Acción activada que contiene una referencia de función o procedimiento" en la página 348

Acciones activadas calificadas por condiciones

La *condición de la acción activada* es una cláusula opcional de la acción activada que especifica una condición de búsqueda que se debe evaluar como *true* para que se ejecuten las sentencias de SQL contenidas en la acción activada. Si se omite la cláusula WHEN, las sentencias internas de la acción activada se ejecutan siempre.

La condición de la acción activada se evalúa una vez para cada fila, en caso de que el activador sea FOR EACH ROW, y una vez para la sentencia, si el activador es FOR EACH STATEMENT.

Esta cláusula brinda un control adicional que se puede utilizar para afinar las acciones activadas en nombre de un activador. Un ejemplo de la utilidad de la cláusula WHEN es para imponer una regla dependiente de los datos en la que una acción activada sólo se active si el valor entrante queda dentro o fuera de un determinado rango.

Conceptos relacionados:

- "Acción activada" en la página 346
- "Acción activada compuesta por sentencias de SQL" en la página 347
- "Acción activada que contiene una referencia de función o procedimiento" en la página 348

Acción activada compuesta por sentencias de SQL

El conjunto de sentencias de SQL activadas conlleva las acciones reales ocasionadas por la activación de un activador. No todas las operaciones de SQL son significativas en todos los activadores. Según si el tiempo de activación del activador es BEFORE o AFTER, pueden ser adecuadas distintas clases de operaciones como sentencia de SQL activada.

En la mayoría de casos, si cualquier sentencia de SQL activada devuelve un código de retorno negativo, se retrotraen la sentencia de SQL activadora junto con todos los activadores y las acciones de restricción referencial, y se devuelve un error: SQLCODE -723 (SQLSTATE 09000). Se devuelven el nombre del activador, SQLCODE, SQLSTATE y muchas de las señales de la sentencia de SQL activada que ha fallado. Las condiciones de error que se producen si se ejecutan activadores que son críticos o se retrotrae la unidad de trabajo completa no se devuelven utilizando SQLCODE -723 (SQLSTATE 09000).

La sentencia de SQL activada de todos los activadores puede ser una sentencia dinámica compuesta. Es decir, pueden contener uno de los elementos siguientes o más:

- sentencia DECLARE variable
- sentencia SET variable
- WHILE bucle
- FOR bucle
- sentencia IF
- sentencia SIGNAL
- sentencia ITERATE
- sentencia LEAVE
- sentencia GET DIGNOSTIC
- selección completa

No obstante, sólo los activadores AFTER e INSTEAD OF pueden contener uno de los elementos siguientes o más:

- sentencia UPDATE de SQL
- sentencia DELETE de SQL
- sentencia INSERT de SQL

Conceptos relacionados:

- "Acción activada" en la página 346
- "Acciones activadas calificadas por condiciones" en la página 346
- "Acción activada que contiene una referencia de función o procedimiento" en la página 348

Acción activada que contiene una referencia de función o procedimiento

Los procedimientos y las funciones, incluidas las funciones definidas por el usuario (UDF), se pueden invocar desde la acción activada de un activador. Los procedimientos se pueden invocar utilizando la sentencia CALL. Las funciones se pueden invocar dentro de cualquier sentencia de SQL activada. La invocación de una rutina desde un activador permite que éste contenga una lógica compleja. Tome en consideración el ejemplo siguiente, que muestra la definición de un activador que contiene la invocación de un procedimiento de SQL denominado TOTAL_SALES:

```
CREATE TRIGGER trig1 AFTER UPDATE ON t1
REFERENCING NEW AS n
FOR EACH ROW MODE DB2SQL
WHEN (n.c1 > 100);
BEGIN ATOMIC
   DECLARE rs INTEGER DEFAULT 0;
   CALL TOTAL_SALES(n.c1, n.c2);
   GET DIANOSTICS rs = RETURN_STATUS;
   VALUES(CASE WHEN rc < 0
               THEN RAISE_ERROR('70001',
                      'PROC CALL failed'));
END;
```

El procedimiento se puede considerar como una subrutina para el activador. Después de que se haya invocado el procedimiento de SQL, el estado de retorno del mismo se comprueba ejecutando la sentencia GET DIAGNOSTICS. Se emite un error si el estado de retorno indica que se ha producido un error en el procedimiento.

A continuación, se muestra un ejemplo de una referencia de función dentro del cuerpo de un activador. Se hace referencia a la función dentro de la cláusula VALUES.

```
CREATE TRIGGER REORDER
AFTER UPDATE OF ON_HAND, MAX_STOCKED ON PARTS
REFERENCING NEW AS N_ROW
FOR EACH ROW
WHEN (N_ROW.ON_HAND < 0.10 * N_ROW.MAX_STOCKED)
 BEGIN ATOMIC
  VALUES (ISSUE_SHIP_REQUEST
          (N_ROW.MAX_STOCKED - N_ROW.ON_HAND, N_ROW.PARTNO));
 END
```

La función ISSUE_SHIP_REQUEST puede ser una función externa que envía un mensaje de correo electrónico al departamento de transporte para notificar que se requiere el pedido de una pieza. La función toma una expresión que contiene variables de transición como parámetro.

Cuando una acción activada contiene una llamada de procedimiento con un nombre de procedimiento no calificado o una sentencia de SQL de acción activada que contiene una referencia de función con un nombre de función no calificado, el procedimiento o la función se resuelve basándose en lo siguiente:

- la vía de acceso de SQL en el momento de creación del activador.
- el privilegio EXECUTE sobre la rutina que mantiene el creador del activador.

Las rutinas se pueden escribir en SQL, Java™, C, C++ o en un lenguaje .NET. Esto permite un control complejo de los flujos de operaciones lógicas, del manejo y la recuperación de errores, y del acceso a las funciones del sistema y de la biblioteca.

Esta posibilidad permite que una acción activada realice operaciones de tipo distinto al de SQL cuando se activa un activador. Por ejemplo, una UDF llamada desde un activador puede enviar un mensaje de correo electrónico y, por ello, actuar como un mecanismo de alerta. Las acciones externas, tales como los mensajes, no están bajo el control de una confirmación y se ejecutarán independientemente del éxito o fracaso del resto de acciones activadas.

Asimismo, la función puede devolver un SQLSTATE que indique que se ha producido un error que ha tenido como consecuencia que fallara la sentencia de SQL activadora. Éste es un método para implementar restricciones definidas por el usuario. (El otro sería la utilización de una sentencia SIGNAL SQLSTATE). Para poder utilizar un activador como medio para comprobar restricciones complejas definidas por el usuario, puede utilizar la función incorporada RAISE_ERROR en una sentencia de SQL activada. Se puede utilizar esta función para devolver a las aplicaciones un SQLSTATE (SQLCODE -438) definido por el usuario.

Por ejemplo, considere algunas reglas relativas a la columna HIREDATE de la tabla EMPLOYEE, donde HIREDATE es la fecha en que el empleado empieza a trabajar.
- HIREDATE debe ser la fecha de inserción o una fecha futura
- HIREDATE no puede ser más tarde de 1 año a partir de la fecha de inserción.
- Si HIREDATE está entre 6 y 12 meses desde la fecha de inserción, notifíquelo al director de personal utilizando una UDF denominada send_note.

El activador siguiente maneja todas estas reglas de INSERT:

```
CREATE TRIGGER CHECK_HIREDATE
  NO CASCADE BEFORE INSERT ON EMPLOYEE
  REFERENCING NEW AS NEW_EMP
 FOR EACH ROW
  BEGIN ATOMIC
  VALUES CASE
    WHEN NEW_EMP.HIREDATE - CURRENT DATE > 600.
      AND NEW_EMP.HIREDATE - CURRENT DATE <eq; 10000.
      THEN SEND_NOTE('persmgr', NEW_EMP.EMPNO, 'late.txt')
    WHEN NEW_EMP.HIREDATE < CURRENT DATE
      THEN RAISE_ERROR('85001', 'HIREDATE has passed')
    WHEN NEW_EMP.HIREDATE - CURRENT DATE > 10000.
      THEN RAISE_ERROR('85002', 'HIREDATE too far out')
    END;
  END
```

Conceptos relacionados:
- “Acción activada” en la página 346
- “Acciones activadas calificadas por condiciones” en la página 346
- “Acción activada compuesta por sentencias de SQL” en la página 347
- “Invocación de la rutina” en la página 209

Tareas relacionadas:
- “Invocación de funciones de tabla definidas por el usuario” en la página 230
- “Llamada a procedimientos desde activadores o rutinas de SQL” en la página 219

Activadores múltiples

Cuando los activadores se definen utilizando la sentencia CREATE TRIGGER, el tiempo de creación de los mismos se registra en la base de datos en forma de indicación de la hora. El valor de esta indicación de la hora se utiliza posteriormente para ordenar la activación de activadores cuando se debe ejecutar a la vez más de un activador. Por ejemplo, se utiliza la indicación de la hora cuando existe más de un activador sobre la misma tabla sujeto, con el mismo suceso y el mismo tiempo de activación. También se utiliza la indicación de la hora cuando hay uno o más activadores AFTER o INSTEAD OF que son activados por el suceso activador y las acciones de restricción referencial ocasionadas directa o indirectamente (es decir, recurrentemente por otras restricciones referenciales) por la acción activada.

Considere los dos activadores siguientes:

```
CREATE TRIGGER NEW_HIRED
   AFTER INSERT ON EMPLOYEE
FOR EACH ROW
   BEGIN ATOMIC
    UPDATE COMPANY_STATS
    SET NBEMP = NBEMP + 1;
  END

CREATE TRIGGER NEW_HIRED_DEPT
   AFTER INSERT ON EMPLOYEE
   REFERENCING NEW AS EMP
FOR EACH ROW
   BEGIN ATOMIC
       UPDATE DEPTS
       SET NBEMP = NBEMP + 1
       WHERE DEPT_ID = EMP.DEPT_ID;
     END
```

Los activadores anteriores se activan cuando se ejecuta una operación INSERT sobre la tabla de empleados. En este caso, la indicación de la hora de creación de los mismos define cuál de los dos activadores anteriores se activará en primer lugar.

La activación de los activadores se lleva a cabo en orden ascendente del valor de indicación de la hora. Así, un activador recién añadido a una base de datos se ejecuta después que todos los otros activadores que se habían definido previamente.

Los activadores antiguos se activan antes que los nuevos para asegurarse de que los activadores nuevos se pueden utilizar como adiciones *incrementales* a los cambios que afectan a la base de datos. Por ejemplo, si una sentencia de SQL activada por el activador T1 inserta una nueva fila en la tabla T, una sentencia de SQL activada por el activador T2 que se ejecute después que T1 se puede utilizar para actualizar la misma fila de T con valores específicos. Activando los activadores en orden ascendente de creación, se puede cerciorar de que las acciones de los nuevos activadores se ejecuten sobre una base de datos que refleje el resultado de la activación de todos los activadores antiguos.

Conceptos relacionados:

- "Activadores en el desarrollo de aplicaciones" en la página 331

Tareas relacionadas:

- "Extracción de información de los UDT, las UDF y los LOB con activadores" en la página 351
- "Prevención de operaciones sobre tablas mediante activadores" en la página 352
- "Definición de reglas empresariales mediante activadores" en la página 353
- "Definición de acciones mediante activadores" en la página 353

Información relacionada:

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

Sinergia entre activadores, restricciones y rutinas

Extracción de información de los UDT, las UDF y los LOB con activadores

Se podría escribir una aplicación que almacenara mensajes de correo electrónico completos como un valor LOB dentro de la columna MESSAGE de la tabla ELECTRONIC_MAIL. Para manipular el correo electrónico, se podría utilizar un procedimiento almacenado o una UDF a fin de extraer información de la columna de mensajes cada vez que dicha información fuera necesaria dentro de una sentencia de SQL.

Observe que las consultas no extraen información ni la almacenan explícitamente como columnas de tablas. Si se hiciera así, se aumentaría el rendimiento de las consultas, no sólo porque no se invoca repetidamente el procedimiento almacenado o la UDF, sino también porque entonces se pueden definir índices en la información extraída.

Mediante la utilización de activadores, se puede extraer esta información siempre que se almacene correo electrónico nuevo en la base de datos. Para hacerlo, defina un activador BEFORE para extraer la información correspondiente de la forma siguiente:

```
CREATE TRIGGER EXTRACT_INFO
  NO CASCADE BEFORE INSERT ON ELECTRONIC_MAIL
  REFERENCING NEW AS N
 FOR EACH ROW
  BEGIN ATOMIC
    SET (N.SENDER, N.RECEIVER, N.SENT_ON, N.SUBJECT)
      = (SELECT SENDER, RECEIVER, SENT_ON, SUBJECT FROM
         TABLE(EMAIL_HEADER(N.MESSAGE)) AS H)
  END
```

Se puede hacer lo mismo añadiendo columnas generadas a la tabla ELECTRONIC_MAIL.

```
ALTER TABLE ELECTRONIC_MAIL
  ADD COLUMN SENDER VARCHAR(200) GENERATED ALWAYS
    AS (SENDER(N.MESSAGE))
  ADD COLUMN RECEIVER VARCHAR(200) GENERATED ALWAYS
    AS (RECEIVER(N.MESSAGE))
  ADD COLUMN SENT_ON DATE GENERATED ALWAYS
    AS (SENDING_DATE(N.MESSAGE))
  ADD COLUMN SUBJECT VARCHAR(200) GENERATED ALWAYS
    AS (SUBJECT(N.MESSAGE))
```

Ahora, siempre que se inserte correo electrónico nuevo en la columna MESSAGE, se extraerán del mensaje el remitente, el receptor, la fecha en que se ha enviado y el asunto del mismo, y se almacenarán en columnas separadas.

Conceptos relacionados:

- "Acción activada" en la página 346
- "Acciones activadas calificadas por condiciones" en la página 346
- "Acción activada compuesta por sentencias de SQL" en la página 347
- "Acción activada que contiene una referencia de función o procedimiento" en la página 348
- "Activadores múltiples" en la página 350

Tareas relacionadas:

- "Prevención de operaciones sobre tablas mediante activadores" en la página 352
- "Definición de reglas empresariales mediante activadores" en la página 353
- "Definición de acciones mediante activadores" en la página 353

Información relacionada:

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

Prevención de operaciones sobre tablas mediante activadores

Suponga que desea evitar que el correo que envíe, que no se haya entregado y se le haya devuelto (tal vez a causa de que la dirección de correo era incorrecta), se almacene en la tabla del correo electrónico.

Para ello, tendrá que evitar la ejecución de determinadas sentencias INSERT de SQL. Existen dos maneras de hacerlo:

- Definir un activador BEFORE que emita un error siempre que el asunto de un correo electrónico sea *correo no entregado*:

```
CREATE TRIGGER BLOCK_INSERT
  NO CASCADE BEFORE INSERT ON ELECTRONIC_MAIL
  REFERENCING NEW AS N
  FOR EACH ROW
  WHEN (SUBJECT(N.MESSAGE) = 'undelivered mail')
  BEGIN ATOMIC
    SIGNAL SQLSTATE '85101'
      SET MESSAGE_TEXT = ('Attempt to insert undelivered mail');
    END
```

- Definir una restricción de comprobación que imponga que los valores de asunto de la nueva columna no sean *correo no entregado*:

```
ALTER TABLE ELECTRONIC_MAIL
  ADD CONSTRAINT NO_UNDELIVERED
  CHECK (SUBJECT <> 'undelivered mail')
```

Debido a las ventajas de la naturaleza declarativa de las restricciones, generalmente se debe definir la restricción en lugar del activador.

Conceptos relacionados:

- "Activadores múltiples" en la página 350
- "Activadores en el desarrollo de aplicaciones" en la página 331

Tareas relacionadas:

- "Extracción de información de los UDT, las UDF y los LOB con activadores" en la página 351
- "Definición de reglas empresariales mediante activadores" en la página 353
- "Definición de acciones mediante activadores" en la página 353

Información relacionada:

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

Definición de reglas empresariales mediante activadores

Suponga que en la empresa existe la política de que en todo el correo electrónico que trate de quejas de clientes aparezca el Sr. Nelson, director comercial, en la lista de copias (CC). Debido a esta regla, es posible que se desee expresar esta condición como restricción, como una de las siguientes (suponiendo la existencia de una UDF CC_LIST para comprobarla):

```
ALTER TABLE ELECTRONIC_MAIL ADD
  CHECK (SUBJECT <> 'Customer complaint' OR
         CONTAINS (CC_LIST(MESSAGE), 'nelson@vnet.ibm.com') = 1)
```

Sin embargo, una restricción de este tipo impide la inserción de correo electrónico que trate de quejas de clientes que no contengan al director comercial en la lista de CC. Ciertamente, ésta no es la intención de la regla empresarial. Lo que se pretende es remitir al director comercial todo el correo electrónico, que trate de quejas de clientes, del que no se haya enviado copia al director comercial. Este tipo de regla empresarial sólo se puede expresar con un activador, pues requiere que se emprendan acciones que no se pueden expresar con restricciones declarativas. El activador supone la existencia de una función SEND_NOTE con parámetros de los tipos E_MAIL y serie de caracteres.

```
CREATE TRIGGER INFORM_MANAGER
  AFTER INSERT ON ELECTRONIC_MAIL
  REFERENCING NEW AS N
FOR EACH ROW
  WHEN (N.SUBJECT = 'Customer complaint' AND
    CONTAINS (CC_LIST(MESSAGE), 'nelson@vnet.ibm.com') = 0)
  BEGIN ATOMIC
    VALUES(SEND_NOTE(N.MESSAGE, 'nelson@vnet.ibm.com'));
  END
```

Conceptos relacionados:

- "Activadores múltiples" en la página 350
- "Activadores en el desarrollo de aplicaciones" en la página 331

Tareas relacionadas:

- "Extracción de información de los UDT, las UDF y los LOB con activadores" en la página 351
- "Prevención de operaciones sobre tablas mediante activadores" en la página 352
- "Definición de acciones mediante activadores" en la página 353

Definición de acciones mediante activadores

Suponga que el director general desea conservar en una tabla separada los nombres de los clientes que han enviado tres o más quejas en las 72 últimas horas. Asimismo, el director general desea ser informado siempre que el nombre de un cliente se inserte más de una vez en esta tabla.

Para indicar estas acciones, defina lo siguiente:

- Una tabla UNHAPPY_CUSTOMERS:

```
CREATE TABLE UNHAPPY_CUSTOMERS (
  NAME           VARCHAR (30),
  EMAIL_ADDRESS  VARCHAR (200),
  INSERTION_DATE DATE)
```

- Un activador para insertar automáticamente una fila en UNHAPPY_CUSTOMERS si en los 3 últimos días se han recibido 3 o más mensajes (suponiendo la existencia de una tabla CUSTOMERS que incluya una columna NAME y una columna E_MAIL_ADDRESS):

```
CREATE TRIGGER STORE_UNHAPPY_CUST
  AFTER INSERT ON ELECTRONIC_MAIL
  REFERENCING NEW AS N
  FOR EACH ROW MODE DB2SQL
  WHEN (3 <= (SELECT COUNT(*)
              FROM ELECTRONIC_MAIL
              WHERE SENDER = N.SENDER
                AND SENDING_DATE(MESSAGE) > CURRENT DATE - 3 DAYS)
       )
  BEGIN ATOMIC
    INSERT INTO UNHAPPY_CUSTOMERS
    VALUES ((SELECT NAME
    FROM CUSTOMERS
    WHERE E_MAIL_ADDRESS = N.SENDER), N.SENDER, CURRENT DATE);
  END
```

- Un activador para enviar una nota al director general si se inserta el mismo cliente más de una vez en UNHAPPY_CUSTOMERS (suponiendo la existencia de una función SEND_NOTE que toma 2 series de caracteres como entrada):

```
CREATE TRIGGER INFORM_GEN_MGR
  AFTER INSERT ON UNHAPPY_CUSTOMERS
  REFERENCING NEW AS N
FOR EACH ROW
  WHEN (1 <(SELECT COUNT(*)
            FROM UNHAPPY_CUSTOMERS
            WHERE EMAIL_ADDRESS = N.EMAIL_ADDRESS)
       )
  BEGIN ATOMIC
    VALUES(SEND_NOTE('Check customer:' CONCAT N.NAME,
                     'bigboss@vnet.ibm.com'));
  END
```

Conceptos relacionados:

- “Activadores múltiples” en la página 350
- “Activadores en el desarrollo de aplicaciones” en la página 331

Tareas relacionadas:

- “Extracción de información de los UDT, las UDF y los LOB con activadores” en la página 351
- “Prevención de operaciones sobre tablas mediante activadores” en la página 352
- “Definición de reglas empresariales mediante activadores” en la página 353

Información relacionada:

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

Parte 3. Apéndices

Apéndice A. Rutinas DB2GENERAL

Rutinas DB2GENERAL 357
UDF DB2GENERAL 358
Tipos de datos de SQL soportados en rutinas DB2GENERAL 360
Clases de Java para rutinas DB2GENERAL . . . 362
Clases de Java para rutinas DB2GENERAL . . 362
Clase DB2GENERAL de Java: COM.IBM.db2.app.StoredProc 362
Clase DB2GENERAL de Java: COM.IBM.db2.app.UDF 364
Clase DB2GENERAL de Java: COM.IBM.db2.app.Lob 366
Clase DB2GENERAL de Java: COM.IBM.db2.app.Blob 367
Clase DB2GENERAL de Java: COM.IBM.db2.app.Clob 367

Rutinas DB2GENERAL

Las rutinas PARAMETER STYLE DB2GENERAL están escritas en Java™. La creación de rutinas de DB2GENERAL es muy parecida a la creación de rutinas en otros lenguajes de programación soportados. Una vez que las haya creado y registrado, las podrá llamar desde programas escritos en cualquier lenguaje. Normalmente, puede llamar a las API de JDBC desde los procedimientos almacenados, pero no desde las UDF.

Cuando cree rutinas en Java, se recomienda firmemente que las registre utilizando la cláusula PARAMETER STYLE JAVA en la sentencia CREATE. PARAMETER STYLE DB2GENERAL sigue disponible para permitir la implantación de las características siguientes en las rutinas Java:

- funciones de tabla
- áreas reutilizables
- acceso a la estructura DBINFO
- la posibilidad de efectuar una FINAL CALL (y una primera llamada separada) a la función o al método

Si tiene rutinas PARAMETER STYLE DB2GENERAL que no utilizan ninguna de las características anteriores, es recomendable migrarlas a PARAMETER STYLE JAVA para su portabilidad.

Conceptos relacionados:

- "UDF DB2GENERAL" en la página 358
- "Rutinas Java" en la página 181
- "Modelo de ejecución de funciones de tabla para Java" en la página 65

Información relacionada:

- "Tabla de depuración de Java DB2DBG.ROUTINE_DEBUG" en la página 193
- "Administración de los archivos JAR en el servidor de bases de datos" en la página 187
- "Tipos de datos de SQL soportados en rutinas DB2GENERAL" en la página 360
- "Clases de Java para rutinas DB2GENERAL" en la página 362
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.StoredProc" en la página 362
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.UDF" en la página 364
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.Lob" en la página 366
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.Blob" en la página 367
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.Clob" en la página 367

UDF DB2GENERAL

Puede crear y utilizar UDF en Java™ tal como lo haría en otros lenguajes, con sólo unas cuantas diferencias menores en comparación con las UDF en C. Después de codificar la UDF, regístrela con la base de datos. Luego le podrá hacer referencia en las aplicaciones.

En general, si declara una UDF que toma argumentos de los tipos de SQL *t1*, *t2* y *t3*, devolviendo el tipo *t4*, se la llamará como método Java con la signatura de Java esperada:

```
public void nombre ( T1 a, T2 b, T3 c, T4 d)  { .....}
```

Donde:
- *nombre* es el nombre del método Java
- De *T1* a *T4* son los tipos de Java que corresponden a los tipos de SQL que van de *t1* a *t4*.
- *a*, *b* y *c* son nombres de variable para los argumentos de entrada.
- *d* es un nombre de variable que representa el argumento de salida.

Por ejemplo, dada una UDF denominada `sample!test3` que devuelve INTEGER y toma argumentos de tipo CHAR(5), BLOB(10K) y DATE, DB2® espera que la implantación en Java de la UDF tenga la signatura siguiente:

```
import COM.ibm.db2.app.*;
public class sample extends UDF {
   public void test3(String arg1, Blob arg2, String arg3,
                     int result) { ... }
}
```

Las rutinas de Java que implantan funciones de tabla requieren más argumentos. Junto a las variables que representan la entrada, en la fila resultante aparece una variable adicional para cada columna. Por ejemplo, una función de tabla se puede declarar como:

```
public void test4(String arg1, int result1,
                  Blob result2, String  result3);
```

Los valores NULL de SQL se representan mediante variables Java que no se inicializan. Estas variables tienen el valor cero si son de tipo primitivo y un valor nulo de Java si son tipos de objeto, en consonancia con las normas de Java. Para indicar un nulo (NULL) de SQL aparte de un cero corriente, puede llamar a la función `isNull` para cualquier argumento de entrada:

```
{ ....
   if (isNull(1)) { /* el argumento #1 era un NULL de SQL */ }
   else           { /* no NULL */ }
}
```

En el ejemplo anterior, los números de argumento empiezan por el uno. La función `isNull()`, al igual que las otras funciones que siguen, se hereda de la clase `COM.ibm.db2.app.UDF`.

Para devolver un resultado de una UDF escalar o de tabla, utilice el método `set()` en la UDF, de la manera siguiente:

```
{ ....
   set(2, valor);
}
```

Donde '2' es el índice de un argumento de salida y valor es un literal o una variable de un tipo compatible. El número de argumento es el índice en la lista de argumentos de la salida seleccionada. En el primer ejemplo de este apartado, la variable `int result` tiene el índice 4; en el segundo, de `result1` a `result3` tienen los índices de 2 a 4.

Al igual que sucede con los módulos C utilizados en las UDF y los procedimientos almacenados, no se pueden utilizar las corrientes de E/S estándar Java (`System.in`, `System.out` y `System.err`) en las rutinas Java.

Recuerde que todos los archivos de clases Java (o los JAR que contienen las clases) que utilice para implantar una rutina deben residir en el directorio `sqllib/function` o en un directorio especificado en la variable CLASSPATH del gestor de bases de datos.

Normalmente, DB2 llama a una UDF muchas veces, una vez por cada fila de un conjunto de entrada o de resultado de una consulta. Si se especifica SCRATCHPAD en la sentencia CREATE FUNCTION de la UDF, DB2 reconoce que es necesaria cierta "continuidad" entre invocaciones sucesivas de la UDF y, por consiguiente, no se crea la instancia de implantación de la clase de Java para cada llamada, sino que generalmente se hace así una vez por cada referencia a la UDF por sentencia. Generalmente, se crea una instancia antes de la primera llamada y se utiliza después, pero es posible que se creen instancias de las funciones de tabla con más frecuencia. Sin embargo, si se especifica NO SCRATCHPAD para una UDF, ya sea una función escalar o de tabla, se crea una instancia limpia para cada llamada a la UDF.

Un área reutilizable puede resultar útil para guardar información a lo largo de las llamadas a una UDF. Mientras que las UDF de Java y OLE pueden utilizar variables de instancias o establecer el área reutilizable para lograr una continuidad entre llamadas, las UDF de C y C++ deben utilizar el área reutilizable. Las UDF de Java acceden al área reutilizable mediante los métodos getScratchPad() y setScratchPad() disponibles en `COM.ibm.db2.app.UDF`.

Para las funciones de tabla de Java que utilicen un área reutilizable, controle cuándo se debe obtener una nueva instancia del área reutilizable mediante las opciones FINAL CALL o NO FINAL CALL de la sentencia CREATE FUNCTION.

La posibilidad de lograr una continuidad entre llamadas a una UDF por medio de un área reutilizable se controla mediante las opciones SCRATCHPAD y NO SCRATCHPAD de CREATE FUNCTION, independientemente de si se utilizan el área reutilizable de DB2 o variables de instancia.

Para las funciones escalares, se utiliza la misma instancia para la sentencia entera.

Observe que cada referencia a una UDF de Java en una consulta se trata de forma independiente, aunque se haga referencia a la misma UDF varias veces. Esto es lo mismo que sucede con las UDF de OLE, C y C++. Al final de una consulta, si se especifica la opción FINAL CALL para una función escalar, se llama al método close() del objeto. Para las funciones de tabla, siempre se invocará el método close(), tal como se indica en el subapartado que sigue a éste. Si no se define un método close() para la clase UDF, una función stub toma el control y se ignora el suceso.

Si se especifica la cláusula ALLOW PARALLEL para una UDF de Java en la sentencia CREATE FUNCTION, DB2 puede elegir evaluar la UDF en paralelo. Si

así sucede, se pueden crear varios objetos Java diferenciados en distintas particiones. Cada objeto recibe un subconjunto de las filas.

Al igual que con otras UDF, las UDF de Java pueden ser protegidas (FENCED) o no protegidas (NOT FENCED). Las UDF NOT FENCED se ejecutan dentro del espacio de direcciones del mecanismo de bases de datos; las UDF FENCED se ejecutan en un proceso separado. Aunque las UDF de Java no pueden corromper inadvertidamente el espacio de direcciones del proceso que las incorpora, pueden interrumpir o ralentizar el proceso. Por consiguiente, cuando depure UDF escritas en Java, las debe ejecutar como UDF FENCED.

Conceptos relacionados:

- "Rutinas DB2GENERAL" en la página 357
- "Rutinas Java" en la página 181
- "Modelo de ejecución de funciones de tabla para Java" en la página 65

Información relacionada:

- "Tabla de depuración de Java DB2DBG.ROUTINE_DEBUG" en la página 193
- "Tipos de datos de SQL soportados en rutinas DB2GENERAL" en la página 360
- "Clases de Java para rutinas DB2GENERAL" en la página 362
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.StoredProc" en la página 362
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.UDF" en la página 364
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.Lob" en la página 366
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.Blob" en la página 367
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.Clob" en la página 367

Ejemplos relacionados:

- "UDFsqlsv.java -- Provide UDFs to be called by UDFsqlcl.java (JDBC)"
- "UDFsrv.java -- Provide UDFs to be called by UDFcli.java (JDBC)"
- "UDFsrv.java -- Provide UDFs to be called by UDFcli.sqlj (SQLj)"

Tipos de datos de SQL soportados en rutinas DB2GENERAL

Cuando se llama a rutinas con PARAMETER STYLE DB2GENERAL, DB2 convierte los tipos de SQL en tipos de Java, y viceversa, en nombre del usuario. En el paquete de Java `COM.ibm.db2.app` se proporcionan varias de estas clases.

Tabla 37. Tipos de SQL y objetos Java de DB2

Tipo de columna de SQL	Tipo de datos de Java
SMALLINT	short
INTEGER	int
BIGINT	long
REAL[1]	float
DOUBLE	double
DECIMAL(p,s)	java.math.BigDecimal
NUMERIC(p,s)	java.math.BigDecimal
CHAR(*n*)	java.lang.String
CHAR(*n*) FOR BIT DATA	COM.ibm.db2.app.Blob

Tabla 37. Tipos de SQL y objetos Java de DB2 (continuación)

Tipo de columna de SQL	Tipo de datos de Java
VARCHAR(*n*)	java.lang.String
VARCHAR(*n*) FOR BIT DATA	COM.ibm.db2.app.Blob
LONG VARCHAR	java.lang.String
LONG VARCHAR FOR BIT DATA	COM.ibm.db2.app.Blob
GRAPHIC(*n*)	java.lang.String
VARGRAPHIC(*n*)	String
LONG VARGRAPHIC[2]	String
BLOB(*n*)[2]	COM.ibm.db2.app.Blob
CLOB(*n*)[2]	COM.ibm.db2.app.Clob
DBCLOB(*n*)[2]	COM.ibm.db2.app.Clob
DATE[3]	String
TIME[3]	String
TIMESTAMP[3]	String

Notas:

1. La diferencia entre REAL y DOUBLE en el SQLDA es el valor de la longitud (4 u 8).
2. Las clases Blob y Clob se proporcionan en el paquete `COM.ibm.db2.app`. Sus interfaces incluyen rutinas para generar una Corriente de entrada y una Corriente de salida para leer y grabar en un Blob, y un Lector y Grabador para un Clob.
3. Los valores de DATE, TIME y TIMESTAMP de SQL utilizan la codificación de series ISO en Java, al igual que para las UDF codificadas en C.

Las instancias de las clases `COM.ibm.db2.app.Blob` y `COM.ibm.db2.app.Clob` representan los tipos de datos LOB (BLOB, CLOB y DBCLOB). Esta clases proporcionan una interfaz limitada para los LOB de lectura pasados como entrada y los LOB de grabación devueltos como salida. La lectura y la grabación de los LOB se producen a través de objetos estándar de corrientes de E/S de Java. Para la clase Blob, las rutinas `getInputStream()` y `getOutputStream()` devuelven un objeto de Corriente de entrada o de Corriente de salida mediante el cual se puede procesar el contenido del BLOB, unos cuantos bytes cada vez. Para un Clob, las rutinas `getReader()` y getWriter() devolverán un objeto Lector o Grabador mediante el cual se puede procesar el contenido del CLOB o del DBCLOB, unos cuantos caracteres cada vez.

Si se devuelve un objeto de este tipo como salida utilizando el método `set()`, se pueden aplicar conversiones de página de códigos para representar los caracteres Unicode de Java en la página de códigos de la base de datos.

Conceptos relacionados:

- "Rutinas DB2GENERAL" en la página 357
- "UDF DB2GENERAL" en la página 358
- "Rutinas Java" en la página 181
- "Modelo de ejecución de funciones de tabla para Java" en la página 65

Información relacionada:

- "Tipos de datos de SQL soportados en Java" en la página 185
- "Clases de Java para rutinas DB2GENERAL" en la página 362

- "Clase DB2GENERAL de Java: COM.IBM.db2.app.StoredProc" en la página 362
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.UDF" en la página 364
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.Lob" en la página 366
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.Blob" en la página 367
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.Clob" en la página 367

Clases de Java para rutinas DB2GENERAL

Clases de Java para rutinas DB2GENERAL

Esta interfaz proporciona la rutina siguiente para captar una conexión JDBC con el contexto de la aplicación que realiza la incorporación:

```
public java.sql.Connection getConnection()
```

Puede utilizar este descriptor de contexto para ejecutar sentencias de SQL. En el archivo `sqllib/samples/java/StoredProc.java` se listan otros métodos de la interfaz `StoredProc`.

Existen cinco clases/interfaces que se pueden utilizar con Procedimientos almacenados o UDF de Java:

- COM.ibm.db2.app.StoredProc
- COM.ibm.db2.app.UDF
- COM.ibm.db2.app.Lob
- COM.ibm.db2.app.Blob
- COM.ibm.db2.app.Clob

Conceptos relacionados:

- "Rutinas DB2GENERAL" en la página 357
- "UDF DB2GENERAL" en la página 358
- "Rutinas Java" en la página 181

Información relacionada:

- "Tipos de datos de SQL soportados en rutinas DB2GENERAL" en la página 360
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.StoredProc" en la página 362
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.UDF" en la página 364
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.Lob" en la página 366
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.Blob" en la página 367
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.Clob" en la página 367

Clase DB2GENERAL de Java: COM.IBM.db2.app.StoredProc

Una clase de Java que contiene métodos pensados para que se llamen como procedimientos almacenados PARAMETER STYLE DB2GENERAL debe ser pública y debe implantar esta interfaz de Java. Debe declarar una clase de este tipo del modo siguiente:

```
public class clase-STP-usuario extends COM.ibm.db2.app.StoredProc{ ... }
```

Sólo puede llamar a métodos heredados de la interfaz `COM.ibm.db2.app.StoredProc` en el contexto del procedimiento almacenado que se está ejecutando actualmente. Por ejemplo, no se pueden utilizar operaciones sobre argumentos LOB ni llamadas

de establecimiento de resultados o estados después de que se devuelva un procedimiento almacenado. De violar esta norma, se emitirá una excepción de Java.

Las llamadas relacionadas con argumentos utilizan un índice de columnas para identificar la columna a la que se hace referencia. Empiezan por 1 para el primer argumento. Todos los argumentos de un procedimiento almacenado PARAMETER STYLE DB2GENERAL se consideran INOUT y, por lo tanto, constituyen tanto entradas como salidas.

La base de datos capta cualquier excepción devuelta desde el procedimiento almacenado y la devuelve al llamante con SQLCODE -4302, SQLSTATE 38501. Una SQLException o SQLWarning de JDBC se maneja de forma especial y pasa literalmente sus propios SQLCODE, SQLSTATE, etc., a la aplicación llamante.

Los métodos siguientes están asociados a la clase `COM.ibm.db2.app.StoredProc`:

```
public StoredProc() [constructor por omisión]
```

La base de datos llama a este constructor antes de la llamada al procedimiento almacenado.

```
public boolean isNull(int) throws Exception
```

Esta función prueba si un argumento de entrada que tiene el índice indicado es un nulo (NULL) de SQL.

```
public void set(int, short) throws Exception
public void set(int, int) throws Exception
public void set(int, double) throws Exception
public void set(int, float) throws Exception
public void set(int, java.math.BigDecimal) throws Exception
public void set(int, String) throws Exception
public void set(int, COM.ibm.db2.app.Blob) throws Exception
public void set(int, COM.ibm.db2.app.Clob) throws Exception
```

Esta función establece el argumento de salida que tiene el índice indicado con el valor proporcionado. El índice tiene que hacer referencia a un argumento de salida válido, el tipo de datos debe coincidir y el valor debe tener una longitud y un contenido aceptables. Las series que contienen caracteres Unicode se tienen que poder representar en la página de códigos de la base de datos. Si se producen errores, se emite una excepción.

```
public java.sql.Connection getConnection() throws Exception
```

Esta función devuelve un objeto JDBC que representa la conexión de la aplicación llamante con la base de datos. Es análogo al resultado de una llamada `SQLConnect()` nula en un procedimiento almacenado C.

Conceptos relacionados:

- “Rutinas DB2GENERAL” en la página 357
- “UDF DB2GENERAL” en la página 358
- “Rutinas Java” en la página 181

Información relacionada:

- “Tipos de datos de SQL soportados en rutinas DB2GENERAL” en la página 360
- “Clases de Java para rutinas DB2GENERAL” en la página 362
- “Clase DB2GENERAL de Java: COM.IBM.db2.app.UDF” en la página 364
- “Clase DB2GENERAL de Java: COM.IBM.db2.app.Lob” en la página 366

- "Clase DB2GENERAL de Java: COM.IBM.db2.app.Blob" en la página 367
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.Clob" en la página 367

Clase DB2GENERAL de Java: COM.IBM.db2.app.UDF

Una clase de Java que contiene métodos destinados a ser llamados por UDF con PARAMETER STYLE DB2GENERAL debe ser pública e implementar esta interfaz Java. Debe declarar una clase de este tipo del modo siguiente:

```
public class clase-UDF-usuario extends COM.ibm.db2.app.UDF{ ... }
```

Sólo se puede llamar a métodos de la interfaz `COM.ibm.db2.app.UDF` en el contexto de la UDF que se esté ejecutando actualmente. Por ejemplo, no se pueden utilizar operaciones sobre argumentos LOB, llamadas de establecimiento de resultados o estados, etc., después de que una UDF vuelve. De violar esta norma, se emitirá una excepción de Java.

Las llamadas relacionadas con argumentos utilizan un índice de columnas para identificar la columna que se está estableciendo. Empiezan por 1 para el primer argumento. Los argumentos de salida se numeran con números más altos que los de entrada. Por ejemplo, una UDF escalar con tres entradas utiliza el índice 4 para la salida.

La base de datos capta cualquier excepción devuelta por la UDF y la devuelve al llamante con SQLCODE -4302, SQLSTATE 38501.

Los métodos siguientes están asociados a la clase `COM.ibm.db2.app.UDF`:

```
public UDF() [constructor por omisión]
```

La base de datos llama a este constructor al principio de una serie de llamadas a UDF. Precede a la primera llamada a la UDF.

```
public void close()
```

La base de datos llama a esta función al final de la evaluación de una UDF, en caso de que la UDF se haya creado con la opción FINAL CALL. Es análogo a la llamada final para una UDF en C. Para las funciones de tabla, se llama a close() después de la llamada CLOSE al método UDF (si se ha codificado o se ha asumido por omisión NO FINAL CALL), o después de la llamada FINAL (si se ha codificado FINAL CALL). Si una clase UDF de Java no implementa esta función, un apéndice no-op manejará e ignorará este suceso.

```
public int getCallType() throws Exception
```

Los métodos UDF de funciones de tabla utilizan getCallType() para averiguar el tipo de llamada de una llamada determinada. Devuelve un valor como los siguientes (se proporcionan definiciones simbólicas para estos valores en la definición de la clase COM.ibm.db2.app.UDF):

- -2 FIRST call
- -1 OPEN call
- 0 FETCH call
- 1 CLOSE call
- 2 FINAL call

```
public boolean isNull(int) throws Exception
```

Esta función prueba si un argumento de entrada que tiene el índice indicado es un nulo (NULL) de SQL.

```
public boolean needToSet(int) throws Exception
```

Esta función prueba si es necesario establecer un argumento de salida con el índice indicado. Esto puede ser falso para una UDF de tabla declarada con DBINFO, en caso de que el llamante de la UDF no utilice esa columna.

```
public void set(int, short) throws Exception
public void set(int, int) throws Exception
public void set(int, double) throws Exception
public void set(int, float) throws Exception
public void set(int, java.math.BigDecimal) throws Exception
public void set(int, String) throws Exception
public void set(int, COM.ibm.db2.app.Blob) throws Exception
public void set(int, COM.ibm.db2.app.Clob) throws Exception
```

Esta función establece el argumento de salida que tiene el índice indicado con el valor proporcionado. El índice tiene que hacer referencia a un argumento de salida válido, el tipo de datos debe coincidir y el valor debe tener una longitud y un contenido aceptables. Las series que contienen caracteres Unicode se tienen que poder representar en la página de códigos de la base de datos. Si se producen errores, se emite una excepción.

```
public void setSQLstate(String) throws Exception
```

Se puede llamar a esta función desde una UDF para establecer el SQLSTATE que esta llamada devolverá. Una UDF de tabla debe llamar a esta función con ″02000″ para indicar una condición de fin de tabla. Si la serie no es aceptable como SQLSTATE, se emitirá una excepción.

```
public void setSQLmessage(String) throws Exception
```

Esta función es parecida a la función `setSQLstate`. Establece el resultado del mensaje de SQL. Si la serie no es aceptable (por ejemplo, si contiene más de 70 caracteres), se emitirá una excepción.

```
public String getFunctionName() throws Exception
```

Esta función devuelve el nombre de la UDF que se está ejecutando.

```
public String getSpecificName() throws Exception
```

Esta función devuelve el nombre específico de la UDF que se está ejecutando.

```
public byte[] getDBinfo() throws Exception
```

Esta función devuelve una estructura DBINFO pura no procesada para la UDF que se está ejecutando, como matriz de bytes. En primer lugar, se debe declarar con la opción DBINFO.

```
public String getDBname() throws Exception
public String getDBauthid() throws Exception
public String getDBtbschema() throws Exception
public String getDBtbname() throws Exception
public String getDBcolname() throws Exception
public String getDBver_rel() throws Exception
public String getDBplatform() throws Exception
public String getDBapplid() throws Exception
```

Estas funciones devuelven el valor del campo apropiado de la estructura DBINFO de la UDF que se está ejecutando.

```
public int getDBprocid() throws Exception
```

Esta función devuelve el id de rutina del procedimiento que ha invocado, directa o indirectamente, a esta rutina. El id de rutina coincide con la columna ROUTINEID de SYSCAT.ROUTINES que se puede utilizar para recuperar el nombre del procedimiento que realiza la invocación. Si se invoca la rutina que se ejecuta desde una aplicación, getDBprocid() devuelve 0.

```
public int[] getDBcodepg() throws Exception
```

Esta función devuelve el SBCS, el DBCS y los números de página de códigos compuestos para la base de datos, desde la estructura DBINFO. La matriz de enteros devuelta contiene los números respectivos como sus tres primeros elementos.

```
public byte[] getScratchpad() throws Exception
```

Esta función devuelve una copia del cuaderno de apuntes de la UDF que se está ejecutando actualmente. En primer lugar, se debe declarar la UDF con la opción SCRATCHPAD.

```
public void setScratchpad(byte[]) throws Exception
```

Esta función sobregraba el cuaderno de apuntes de la UDF que se está ejecutando actualmente con el contenido de la matriz de bytes indicada. En primer lugar, se debe declarar la UDF con la opción SCRATCHPAD. La matriz de bytes debe tener el mismo tamaño que `getScratchpad()` devuelve.

Conceptos relacionados:

- "Rutinas DB2GENERAL" en la página 357
- "UDF DB2GENERAL" en la página 358
- "Rutinas Java" en la página 181

Información relacionada:

- "Tipos de datos de SQL soportados en rutinas DB2GENERAL" en la página 360
- "Clases de Java para rutinas DB2GENERAL" en la página 362
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.StoredProc" en la página 362
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.Lob" en la página 366
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.Blob" en la página 367
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.Clob" en la página 367

Clase DB2GENERAL de Java: COM.IBM.db2.app.Lob

Esta clase proporciona rutinas de utilidad que crean objetos Blob o Clob temporales para un cálculo interno en las rutinas.

Los métodos siguientes están asociados a la clase COM.ibm.db2.app.Lob:

```
public static Blob newBlob() throws Exception
```

Esta función crea un Blob temporal. Si es posible, se implementará utilizando un LOCALIZADOR.

```
public static Clob newClob() throws Exception
```

Esta función crea un Clob temporal. Si es posible, se implementará utilizando un LOCALIZADOR.

Conceptos relacionados:

- "Rutinas DB2GENERAL" en la página 357
- "UDF DB2GENERAL" en la página 358
- "Rutinas Java" en la página 181

Información relacionada:
- "Tipos de datos de SQL soportados en rutinas DB2GENERAL" en la página 360
- "Clases de Java para rutinas DB2GENERAL" en la página 362
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.StoredProc" en la página 362
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.UDF" en la página 364
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.Blob" en la página 367
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.Clob" en la página 367

Clase DB2GENERAL de Java: COM.IBM.db2.app.Blob

La base de datos pasa una instancia de esta clase para representar un BLOB como entrada de la rutina, y se puede devolver como salida. La aplicación puede crear instancias, pero sólo en el contexto de una rutina en ejecución. Los usos de estos objetos fuera de este tipo de contexto emitirán una excepción.

Los métodos siguientes están asociados a la clase `COM.ibm.db2.app.Blob`:

```
public long size() throws Exception
```

Esta función devuelve la longitud (en bytes) del BLOB.

```
public java.io.InputStream getInputStream() throws Exception
```

Esta función devuelve una nueva Corriente de entrada para leer el contenido del BLOB. Se dispone de operaciones eficaces de búsqueda/marcaje sobre este objeto.

```
public java.io.OutputStream getOutputStream() throws Exception
```

Esta función devuelve una nueva Corriente de salida para añadir bytes al BLOB. Los bytes añadidos pasan a ser visibles de inmediato en todas las instancias de Corriente de entrada existentes producidas por la llamada `getInputStream()` de este objeto.

Conceptos relacionados:
- "Rutinas DB2GENERAL" en la página 357
- "UDF DB2GENERAL" en la página 358
- "Rutinas Java" en la página 181

Información relacionada:
- "Tipos de datos de SQL soportados en rutinas DB2GENERAL" en la página 360
- "Clases de Java para rutinas DB2GENERAL" en la página 362
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.StoredProc" en la página 362
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.UDF" en la página 364
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.Lob" en la página 366
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.Clob" en la página 367

Clase DB2GENERAL de Java: COM.IBM.db2.app.Clob

La base de datos pasa una instancia de esta clase para representar un CLOB o DBCLOB como entrada de la rutina, y se puede devolver como salida. La

aplicación puede crear instancias, pero sólo en el contexto de una rutina en ejecución. Los usos de estos objetos fuera de este tipo de contexto emitirán una excepción.

Las instancias de Clob almacenan caracteres en la página de códigos de la base de datos. Algunos caracteres Unicode no se pueden representar en esta página de códigos y pueden causar que se emita una excepción durante la conversión. Esto se puede producir durante una operación de adición o durante una llamada `set()` de una UDF o de StoredProc. Es necesario para ocultar la distinción entre un CLOB y un DBCLOB del programador Java.

Los métodos siguientes están asociados a la clase `COM.ibm.db2.app.Clob`:

```
public long size() throws Exception
```

Esta función devuelve la longitud (en caracteres) del CLOB.

```
public java.io.Reader getReader() throws Exception
```

Esta función devuelve un nuevo Lector para leer el contenido del CLOB o DBCLOB. Se dispone de operaciones eficaces de búsqueda/marcaje sobre este objeto.

```
public java.io.Writer getWriter() throws Exception
```

Esta función devuelve un nuevo Grabador para añadir caracteres a este CLOB o DBCLOB. Los caracteres añadidos pasan a ser visibles de inmediato en todas las instancias de Lector existentes producidas por la llamada `GetReader()` de este objeto.

Conceptos relacionados:

- "Rutinas DB2GENERAL" en la página 357
- "UDF DB2GENERAL" en la página 358
- "Rutinas Java" en la página 181

Información relacionada:

- "Tipos de datos de SQL soportados en rutinas DB2GENERAL" en la página 360
- "Clases de Java para rutinas DB2GENERAL" en la página 362
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.StoredProc" en la página 362
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.UDF" en la página 364
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.Lob" en la página 366
- "Clase DB2GENERAL de Java: COM.IBM.db2.app.Blob" en la página 367

Apéndice B. Procedimientos COBOL

Procedimientos COBOL . . . . . 369 Tipos de datos de SQL soportados en COBOL . . 371

Procedimientos COBOL

Los procedimientos COBOL se deben escribir de forma similar a los subprogramas COBOL.

Manejo de parámetros en un procedimiento COBOL

Cada parámetro que un procedimiento debe aceptar o pasar se debe declarar en LINKAGE SECTION. Por ejemplo, este fragmento de código corresponde a un procedimiento que acepta dos parámetros IN (un CHAR(15) y un INT) y pasa un parámetro OUT (un INT):

```
LINKAGE SECTION.
01  IN-SPERSON    PIC X(15).
01  IN-SQTY       PIC S9(9)  USAGE COMP-5.
01  OUT-SALESSUM  PIC S9(9)  USAGE COMP-5.
```

Asegúrese de que los tipos de datos COBOL que declare estén correctamente correlacionados con tipos de datos SQL. Para obtener una lista detallada de las correlaciones de tipos de datos entre SQL y COBOL, consulte ″Tipos de datos SQL soportados en COBOL″.

Cada parámetro se deberá listar entonces en PROCEDURE DIVISION. El ejemplo siguiente muestra una PROCEDURE DIVISION que corresponde a las definiciones de parámetros del ejemplo anterior de LINKAGE SECTION.

```
PROCEDURE DIVISION USING IN-SPERSON
                         IN-SQTY
                         OUT-SALESSUM.
```

Cómo salir de un procedimiento COBOL

Para salir correctamente del procedimiento, utilice los mandatos siguientes:

```
MOVE SQLZ-HOLD-PROC TO RETURN-CODE.
          GOBACK.
```

Con estos mandatos, el procedimiento le devuelve correctamente a la aplicación cliente. Esto resulta especialmente importante cuando llama al procedimiento una aplicación cliente COBOL local.

Al crear un procedimiento COBOL, es muy recomendable que utilice el script de creación escrito para su sistema operativo y compilador. Los scripts de creación para Micro Focus COBOL se encuentran en el directorio sqllib/samples/cobol_mf. Los scripts de creación para IBM® COBOL se encuentran en el directorio sqllib/samples/cobol.

A continuación, se muestra un ejemplo de procedimiento COBOL que acepta dos parámetros de entrada y luego devuelve un parámetro de salida y un conjunto de resultados:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.    "NEWSALE".
DATA DIVISION.

WORKING-STORAGE SECTION.
```

```
       01  INSERT-STMT.
           05  FILLER   PIC X(24) VALUE "INSERT INTO SALES (SALES".
           05  FILLER   PIC X(24) VALUE "_PERSON,SALES) VALUES ('".
           05  SPERSON  PIC X(16).
           05  FILLER   PIC X(2) VALUE "',".
           05  SQTY     PIC S9(9).
           05  FILLER   PIC X(1) VALUE ")".
           EXEC SQL BEGIN DECLARE SECTION END-EXEC.
       01  INS-SMT-INF.
           05  INS-STMT.
           49  INS-LEN   PIC S9(4) USAGE COMP.
           49  INS-TEXT  PIC X(100).
       01  SALESSUM      PIC S9(9)  USAGE COMP-5.
           EXEC SQL END DECLARE SECTION END-EXEC.
           EXEC SQL INCLUDE SQLCA END-EXEC.

       LINKAGE SECTION.
       01  IN-SPERSON    PIC X(15).
       01  IN-SQTY       PIC S9(9)  USAGE COMP-5.
       01  OUT-SALESSUM  PIC S9(9)  USAGE COMP-5.

       PROCEDURE DIVISION USING IN-SPERSON
                                IN-SQTY
                                OUT-SALESSUM.
       MAINLINE.
           MOVE 0 TO SQLCODE.
           PERFORM INSERT-ROW.
           IF SQLCODE IS NOT EQUAL TO 0
              GOBACK
           END-IF.
           PERFORM SELECT-ROWS.
           PERFORM GET-SUM.
           GOBACK.
       INSERT-ROW.
           MOVE IN-SPERSON TO SPERSON.
           MOVE IN-SQTY TO SQTY.
           MOVE           INSERT-STMT TO INS-TEXT.
           MOVE LENGTH OF INSERT-STMT TO INS-LEN.
           EXEC SQL EXECUTE IMMEDIATE :INS-STMT END-EXEC.
       GET-SUM.
 EXEC SQL
              SELECT SUM(SALES) INTO :SALESSUM FROM SALES
           END-EXEC.
           MOVE SALESSUM TO OUT-SALESSUM.
       SELECT-ROWS.
 EXEC SQL
              DECLARE CUR CURSOR WITH RETURN FOR SELECT * FROM SALES
           END-EXEC.
           IF SQLCODE = 0
              EXEC SQL OPEN CUR END-EXEC
           END-IF.
```

La sentencia CREATE PROCEDURE que corresponde a este procedimiento es la siguiente:

```
CREATE PROCEDURE NEWSALE ( IN SALESPERSON CHAR(15),
                           IN SALESQTY INT,
                           OUT SALESSUM INT)
  RESULT SETS 1
  EXTERNAL NAME 'NEWSALE!NEWSALE'
  FENCED
  LANGUAGE COBOL
  PARAMETER STYLE SQL
  MODIFIES SQL DATA
```

La sentencia anterior presupone que la función COBOL se halla en una biblioteca denominada NEWSALE.

Nota: Al registrar un procedimiento COBOL en los sistemas operativos Windows®, tome la precaución siguiente cuando identifique un cuerpo de procedimiento almacenado en la cláusula EXTERNAL NAME de la sentencia CREATE. Si utiliza un ID de vía de acceso absoluto para identificar el cuerpo del procedimiento, debe añadir la extensión .dll. Por ejemplo:

```
CREATE PROCEDURE NEWSALE ( IN SALESPERSON CHAR(15),
                           IN SALESQTY INT,
                           OUT SALESSUM INT)
  RESULT SETS 1
  EXTERNAL NAME 'NEWSALE!NEWSALE'
  FENCED
  LANGUAGE COBOL
  PARAMETER STYLE SQL
  MODIFIES SQL DATA
  EXTERNAL NAME 'd:\mylib\NEWSALE.dll'
```

Conceptos relacionados:

- "Procedimientos" en la página 13
- "Sentencias de SQL incorporado en COBOL" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente*
- "Variables del lenguaje principal en COBOL" en la publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente*

Tareas relacionadas:

- "Creación de rutinas IBM COBOL en AIX" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de rutinas Micro Focus COBOL para UNIX" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de rutinas IBM COBOL en Windows" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*
- "Creación de rutinas Micro Focus COBOL en Windows" en la publicación *Guía de desarrollo de aplicaciones: Creación y ejecución de aplicaciones*

Información relacionada:

- "Tipos de datos de SQL soportados en COBOL" en la página 371
- "Sentencia CREATE PROCEDURE (Externo)" en la publicación *Consulta de SQL, Volumen 2*
- "Sintaxis para pasar argumentos a rutinas escritas en C/C++, OLE o COBOL" en la página 97

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 38. 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
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

Tabla 38. 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
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
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 del 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 el 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 publicación *Guía de desarrollo de aplicaciones: Programación de aplicaciones de cliente*

Apéndice C. 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 376

Tareas relacionadas:

- "Invocación de ayuda según contexto desde una herramienta de DB2" en la página 394
- "Actualización del Centro de información de DB2 instalado en el sistema o en un servidor de intranet" en la página 386
- "Invocación de la ayuda de mensajes desde el procesador de línea de mandatos" en la página 396
- "Invocación de la ayuda de mandatos desde el procesador de línea de mandatos" en la página 396
- "Invocación de la ayuda para estados de SQL desde el procesador de línea de mandatos" en la página 397

Información relacionada:

- "Documentación PDF e impresa de DB2" en la página 388

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 378

Tareas relacionadas:

- "Actualización del Centro de información de DB2 instalado en el sistema o en un servidor de intranet" en la página 386
- "Visualización de temas en el idioma preferido en el Centro de información de DB2" en la página 387
- "Invocación del Centro de información de DB2" en la página 385
- "Instalación del Centro de información de DB2 utilizando el asistente de instalación de DB2 (UNIX)" en la página 380
- "Instalación del Centro de información de DB2 utilizando el asistente de instalación de DB2 (Windows)" en la página 383

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 376

Tareas relacionadas:

- "Actualización del Centro de información de DB2 instalado en el sistema o en un servidor de intranet" en la página 386
- "Instalación del Centro de información de DB2 utilizando el asistente de instalación de DB2 (UNIX)" en la página 380
- "Instalación del Centro de información de DB2 utilizando el asistente de instalación de DB2 (Windows)" en la página 383

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 376
- "Escenarios de instalación del Centro de información de DB2" en la página 378

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 386
- "Visualización de temas en el idioma preferido en el Centro de información de DB2" en la página 387
- "Invocación del Centro de información de DB2" en la página 385
- "Instalación del Centro de información de DB2 utilizando el asistente de instalación de DB2 (Windows)" en la página 383

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 376

- "Escenarios de instalación del Centro de información de DB2" en la página 378

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 386
- "Visualización de temas en el idioma preferido en el Centro de información de DB2" en la página 387
- "Invocación del Centro de información de DB2" en la página 385
- "Instalación del Centro de información de DB2 utilizando el asistente de instalación de DB2 (UNIX)" en la página 380

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 376

Tareas relacionadas:

- "Visualización de temas en el idioma preferido en el Centro de información de DB2" en la página 387
- "Invocación de ayuda según contexto desde una herramienta de DB2" en la página 394
- "Actualización del Centro de información de DB2 instalado en el sistema o en un servidor de intranet" en la página 386
- "Invocación de la ayuda de mensajes desde el procesador de línea de mandatos" en la página 396
- "Invocación de la ayuda de mandatos desde el procesador de línea de mandatos" en la página 396
- "Invocación de la ayuda para estados de SQL desde el procesador de línea de mandatos" en la página 397

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 378

Tareas relacionadas:

- "Invocación del Centro de información de DB2" en la página 385
- "Instalación del Centro de información de DB2 utilizando el asistente de instalación de DB2 (UNIX)" en la página 380
- "Instalación del Centro de información de DB2 utilizando el asistente de instalación de DB2 (Windows)" en la página 383

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 376

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 39. 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 40. 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 41. 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 41. 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 42. 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 43. Información de DB2 Connect

Nombre	Número de documento	Nombre de archivo PDF
IBM Connectivity Supplement	Sin número de documento	db2h1x81

Tabla 43. 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 44. 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 45. 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 46. 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 47. 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 375

Tareas relacionadas:

- "Impresión de manuales de DB2 desde archivos PDF" en la página 393
- "Solicitud de manuales de DB2 impresos" en la página 394
- "Invocación de ayuda según contexto desde una herramienta de DB2" en la página 394

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 376

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 394
- "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 388

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 393

Información relacionada:

- "Documentación PDF e impresa de DB2" en la página 388

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 385
- “Invocación de la ayuda de mensajes desde el procesador de línea de mandatos” en la página 396
- “Invocación de la ayuda de mandatos desde el procesador de línea de mandatos” en la página 396
- “Invocación de la ayuda para estados de SQL desde el procesador de línea de mandatos” en la página 397
- “Cómo utilizar la ayuda de DB2 UDB”
- “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 394
- "Invocación del Centro de información de DB2" en la página 385
- "Invocación de la ayuda de mensajes desde el procesador de línea de mandatos" en la página 396
- "Invocación de la ayuda para estados de SQL desde el procesador de línea de mandatos" en la página 397

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 385
- "Invocación de la ayuda de mensajes desde el procesador de línea de mandatos" en la página 396
- "Invocación de la ayuda de mandatos desde el procesador de línea de mandatos" en la página 396

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 376
- "Introduction to problem determination - DB2 Technical Support tutorial" en la publicación *Troubleshooting Guide*

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 400.
- La documentación de DB2 se proporciona en un formato accesible. Si desea más información, consulte el apartado "Documentación accesible" en la página 400.

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 400

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 399

Tareas relacionadas:

- "Contenido"

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 D. 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

.NET
  ejecución en el lenguaje común
    rutinas 116

A

accesibilidad
  características 399
  diagramas de sintaxis decimal con puntos 400
activadores
  acceso a valores de columna en 344
  acción activada compuesta por sentencias de SQL 347
  acciones 353
    calificados por condiciones 346
    compuestas por sentencias de SQL 347
  activación de INSTEAD OF 336, 340
  actualizaciones
    UPDATE, operación 335
  antes de actualizaciones 340
  clasificación de varios 350
  cláusula WHEN 346
  condición de acción activada 346
  creación 338
    directrices para 337
  después de actualizaciones 340
  devolución de SQLSTATE 331
  ejemplos de
    definición de reglas empresariales mediante activadores 353
    ejemplo de una acción de activador 353
  extracción de información de UDT, UDF y LOB con 351
  FOR EACH ROW, cláusula 339
  FOR EACH STATEMENT, cláusula 339
  función RAISE_ERROR 348
  granularidad 339
  INSERT, operación 335
  interacción con restricciones 336
  interacción con restricciones referenciales 336
  secuenciado 350
  sentencias de SQL activadas 346
  supresión 335
  tablas de transición 344
  tiempo de activación 340
  utilización de activadores
    para definir reglas empresariales 353
    prevención de operaciones sobre tablas mediante activadores 352
  variables de transición
    descripción 343
activadores *(continuación)*
  variables de transición *(continuación)*
    nombre de correlación NEW AS 343
    nombre de correlación OLD AS 343
activadores BEFORE
  utilización
    para impedir operaciones sobre las tablas 352
activadores DELETE 343
Actualización
  documentación HMTL 386
ALTER VIEW, sentencia
  tipos estructurados 293
ALLOCATE CURSOR, sentencia
  rutina llamante 53
ámbito
  en tablas con tipo 276
aplicaciones de varias hebras
  rutinas SQLJ 189
archivos
  lectura desde un archivo en un CLOB 245
áreas reutilizables 24
  para UDF y métodos 58
  plataformas de 32 bits y de 64 bits 61
  UDF Java 358
ASSOCIATE RESULT SET LOCATOR, sentencia 53
atajos del teclado
  soporte para 399
auditoría
  transacciones
    utilización de funciones de SQL 90
automatización de OLE
  controladores 194
  identificador de clase (CLSID) 195
  identificador programático (progID) 195
  métodos 194
  rutinas
    definición 195
    instancias de objetos 196
    invocación de métodos 196
    SCRATCHPAD, opción 196
  servidores 194
  tipo de datos BSTR 198
  tipo de datos OLECHAR 198
  tipos de datos string 198
autorización
  para rutinas externas 38
ayuda
  para mandatos
    invocación 396
  para mensajes
    invocación 396
  para sentencias de SQL
    invocación 397
ayuda *(continuación)*
  visualización 385, 387
ayuda para mandatos
  invocación 396
ayuda para mensajes
  invocación 396
ayuda para sentencias de SQL
  invocación 397

B

BASIC, lenguaje 195
BASIC, tipos de datos 198
BigDecimal, tipo de datos de Java 185
BIGINT, tipo de datos
  función de tabla de OLE DB 206
  funciones definidas por el usuario (UDF)
    C/C 171
  rutinas
    Java (DB2GENERAL) 360
BIGINT, tipo de datos de SQL
  COBOL 371
  Java 185
BLOB, tipo de datos
  COBOL 371
  función de tabla de OLE DB 206
  Java 185
  rutinas
    C/C++ 171
    Java (DB2GENERAL) 360
BLOB-FILE, tipo de COBOL 371
BLOB-LOCATOR, tipo de COBOL 371

C

C
  procedimientos almacenados, manejo de parámetros 56
  rutinas
    archivo de inclusión 168
    rendimiento 24
    sintaxis para pasar argumentos 97
    tipos de datos de SQL soportados en 168
C++
  decoración de tipos para cuerpos de rutinas 179
  rutinas
    archivo de inclusión 168
    tipos de datos de SQL soportados en 168
  tipos de datos, automatización de OLE 198
CALL, procedimientos
  desde activadores 219
  desde aplicaciones 217
  desde el Procesador de línea de mandatos (CLP) 222

CALL, procedimientos *(continuación)*
desde rutinas de SQL 219
desde rutinas externas 217
CALL, sentencias
procedimientos almacenados 222
CAST FROM, cláusula
manejo de tipos de datos 171
catálogos del sistema
eliminación
implicaciones de las vistas 293
Centro de Desarrollo
depuración de procedimientos almacenados de Java 189, 191
tabla de depuración 192
valores del entorno 190
Centro de información
instalación 378, 380, 383
Centro de información de DB2 376
invocación 385
CLASSPATH, variable de entorno 186
cláusula ADD METHOD en la sentencia ALTER TYPE 271
cláusula PICTURE (PIC) en tipos de COBOL 371
cláusula USAGE en tipos de COBOL 371
cliente, transformaciones de
conversión de tipos de datos 320
enlace de instancias desde una aplicación cliente 319
implantadas utilizando UDF externas 319
visión general 316
CLOB (objeto grande de caracteres)
ejemplos
grabación de datos de una columna CLOB en un archivo 244
Inserción de datos de un archivo de texto en una columna CLOB 245
tipo de datos
COBOL 371
función de tabla de OLE DB 206
funciones definidas por el usuario (UDF), C/C++ 171
Java 185
rutinas, Java (DB2GENERAL) 360
clob_file, tipo de C/C++
ejemplo de
grabación de datos de una columna CLOB en un archivo 244
CLOB-FILE, tipo de COBOL 371
CLOB-LOCATOR, tipo de COBOL 371
CLP (procesador de línea de mandatos)
carácter de terminación 72
CLR
rutinas
creación 117
ejemplos de procedimientos CLR en C# 130
ejemplos de UDF CLR en C# 152
CLR (ejecución en el lenguaje común)
rutinas 116
columnas de identificador de objetos
denominación 276
columnas de identificador de objetos *(continuación)*
descripción 286
COM.ibm.db2.app.Blob 360, 367
COM.ibm.db2.app.Clob 360, 367
COM.ibm.db2.app.Lob 366
COM.ibm.db2.app.StoredProc 362
COM.ibm.db2.app.UDF 358, 364
coma flotante, parámetro 171
comportamiento de ejecución de DYNAMICRULES 113
comportamiento de invocación de DYNAMICRULES 113
comportamiento del enlace DYNAMICRULES 113
condición de acción activada 346
conjuntos de filas
nombres completamente calificados de OLE DB 205
conjuntos de resultados
de procedimientos almacenados 47
devolución de conjuntos de resultados desde un procedimiento
devolución de conjuntos de resultados desde un procedimiento CLR 125
devolución desde un procedimiento almacenado JDBC 52
devolución desde un procedimiento almacenado SQLJ 51
devolución desde un procedimiento de SQL 49
recepción en aplicaciones y rutinas JDBC 55
recepción en aplicaciones y rutinas SQLJ 54
constantes
comparación con tipos diferenciados 259
CONTAINS SQL, cláusula
rutinas que contienen SQL 110
contextos
establecimiento en aplicaciones de DB2 de varias hebras
rutinas SQLJ 189
correlación de tipos
automatización de OLE
tipos BASIC 198
creación
rutinas
ejecución en el lenguaje común 117
CREATE FUNCTION, sentencia
CAST FROM, cláusula 171
LANGUAGE OLE, cláusula 195
RETURNS, cláusula 171
rutinas de automatización de OLE 195
CREATE TABLE, sentencia
definición de opciones de columnas 276
CREATE TRANSFORM, sentencia
utilización 310
CREATE TRIGGER, sentencia 338
cláusula AFTER 336, 340
cláusula BEFORE 336, 340
cláusula INSTEAD OF 336, 340
CREATE TRIGGER, sentencia *(continuación)*
REFERENCING, cláusula 344
CREATE TYPE, sentencia
REF USING, cláusula 286
tipos estructurados 270
cursores
rutinas 110

CH

CHAR, tipo de datos
COBOL 371
función de tabla de OLE DB 206
funciones definidas por el usuario (UDF) 171
Java 185
rutinas, Java (DB2GENERAL) 360
CHAR FOR BIT DATA, tipo de datos 360

D

DATE, tipo de datos
COBOL 371
función de tabla de OLE DB 206
Java 185
rutinas
Java (DB2GENERAL) 360
DB2, guías de aprendizaje 397
DB2DBG.ROUTINE_DEBUG, tabla de depuración 193
DB2GENERAL, estilo de parámetros para rutinas externas 95
DB2SQL, estilo de parámetros para rutinas externas 95
DBCLOB, tipo de datos
COBOL 371
función de tabla de OLE DB 206
funciones definidas por el usuario (UDF)
C/C 171
Java 185
rutinas
Java (DB2GENERAL) 360
DBCLOB-FILE, tipo de COBOL 371
DBCLOB-LOCATOR, tipo de COBOL 371
dbinfo, argumento
funciones de tabla 63
DBINFO, opción
páginas de códigos 214
de tabla
tabla con tipo 276
DECIMAL, tipo de datos
COBOL 371
función de tabla de OLE DB 206
Java 185
parámetro de rutinas externas 171
rutinas
Java (DB2GENERAL) 360
decoración de tipos
cuerpos de rutinas en C++ 179
definición del comportamiento de DYNAMICRULES 113

depuración
procedimientos almacenados 189
rutinas 42
DEREF, función 294
privilegios requeridos 294
DESCRIBE, sentencia
tipos estructurados 329
deshacer referencia, operadores 287
consultas que utilizan 294
determinación de problemas
guías de aprendizaje 398
información en línea 398
devolución de conjuntos de resultados
desde procedimientos almacenados JDBC 52
desde procedimientos almacenados SQLJ 51
desde procedimientos de SQL 49
diagramas de sintaxis decimal con puntos 400
directorio del archivo de registro para procedimientos de SQL 77
documentación
visualización 385
documentación HTML
actualización 386
double, tipo de datos
programas de Java 185
DOUBLE, tipo de datos
funciones definidas por el usuario (UDF)
C/C 171
rutinas externas 171
DROP, sentencia
tipos estructurados en tablas 279
vistas
tipos estructurados 293
DYNAMICRULES, opción de precompilación/enlace
efectos en el SQL dinámico 113

E

ejecución en el lenguaje común
rutinas 116
área-reutilizable 121
creación 117
ejemplos de funciones CLR en C# 152
ejemplos de procedimientos CLR en C# 130
errores relacionados con 128
parámetros de las rutinas de ejecución en el lenguaje común 121
restricciones 126
tipos de datos de SQL soportados en 120
uso de la estructura Dbinfo 121
ejemplos
SQL dinámico, asignación de tipos 260
tipos diferenciados
asignación de tipo de comparación 261
asignación de tipos de comparación 260
ejemplos *(continuación)*
tipos diferenciados *(continuación)*
comparación con tipos diferenciados 258
comparación con valores constantes 257
en UNION 262
en línea
acceso a la ayuda 394
errores
rutinas
relacionados con rutinas de ejecución en el lenguaje común 128
escritura de rutinas 36
estilos de parámetros para rutinas externas 95
EXECUTE, privilegio
rutinas 38
EXECUTE, sentencia
SQL dinámico 69
EXTERNAL NAME, cláusula
CREATE FUNCTION, sentencia 205

F

FLOAT, tipo de datos
COBOL 371
función de tabla de OLE DB 206
funciones definidas por el usuario (UDF) 171
Java 185
rutinas, Java (DB2GENERAL) 360
función, transformaciones de
implantadas como rutinas incorporadas al SQL 313
pase de parámetros a rutinas externas 314
visión general 311
funciones
algoritmo de selección 225
de tabla
funciones de tabla de SQL que modifican datos de SQL 88
escalares
DEREF 294
TYPE_ID 294
TYPE_NAME 294
TYPE_SCHEMA 294
invocación 224
selección 225
sintaxis de las referencias a 224
funciones con fuente
para tipos diferenciados
creación de UDF con fuente para tipos diferenciados 262
funciones de constructor 274
funciones de SQL
funciones de tabla
que modifican datos de SQL 90
funciones de tabla
funciones de tabla de SQL
funciones de tabla de SQL que modifican datos de SQL 88
funciones de tabla definidas por el usuario 63
modelo de ejecución de Java 65
funciones definidas por el usuario (UDF)
C/C++
argumentos 171
BIGINT, tipo de datos 171
BLOB, tipo de datos 171
CLOB, tipo de datos 171
CHAR, tipo de datos 171
DBCLOB, tipo de datos 171
DOUBLE, tipo de datos 171
FLOAT, tipo de datos 171
INTEGER, tipo de datos 171
LONG VARCHAR, tipo de datos 171
parámetros 171
REAL, tipo de datos 171
SMALLINT, tipo de datos 171
VARCHAR FOR BIT DATA, tipo de datos 171
VARGRAPHIC, tipo de datos 171
de tabla
invocación 230
SQL-result, argumento 63
SQL-result-ind, argumento 63
visión general 17
DETERMINISTIC 58
devolución de datos 171
funciones de tabla de OLE DB 201
guardar el estado 58
infix, notación 225
Java
restricciones de E/S 358
modificador FOR BIT DATA 171
NOT DETERMINISTIC 58
parámetros de fecha 171
reentrantes 58
rutinas 3
SCRATCHPAD, opción 58
UDF de ejecución en el lenguaje común
ejemplos en C# 152
funciones definidas por el usuario (UDF) para tablas
modelo de proceso 63
funciones escalares
modelo de proceso 62
visión general 15

G

GENERAL, estilo de parámetros para rutinas externas 95
GENERAL WITH NULLS, estilo de parámetros para rutinas externas 95
grabación de datos de una columna CLOB en un archivo de texto 244
GRANT, sentencia
emisión sobre jerarquías de tablas 276
granularidad 339
GRAPHIC, parámetro 171
GRAPHIC, tipo de datos
COBOL 371
función de tabla de OLE DB 206
Java 185
rutinas
Java (DB2GENERAL) 360
guías de aprendizaje 397

guías de aprendizaje *(continuación)*
resolución de problemas y determinación de problemas 398

I

identificadores de objetos
creación de restricciones 285
generación automática 283
impresión
archivos PDF 393
incapacidad 399
infix, notación
funciones definidas por el usuario (UDF) 225
INHERIT SELECT PRIVILEGES, cláusula 276
instalación
Centro de información 378, 380, 383
instancias de objetos
rutinas de automatización de OLE 196
Int, tipo de datos de Java 185
INTEGER, tipo de datos
COBOL 371
función de tabla de OLE DB 206
funciones definidas por el usuario (UDF)
C/C 171
Java 185
rutinas
Java (DB2GENERAL) 360
integridad referencial
comparación con las referencias con ámbito 290
invocación
ayuda para mandatos 396
ayuda para mensajes 396
ayuda para sentencias de SQL 397
funciones de tabla definidas por el usuario 230
rutinas 209
UDF 229
IS OF, predicado
restricción de los tipos devueltos mediante 296

J

Java
actualización de clases 187
archivos de clases, situación 186
archivos JAR 187
CLASSPATH, variable de entorno 186
COM.ibm.db2.app. StoredProc 362
COM.ibm.db2.app.Blob 367
COM.ibm.db2.app.Clob 367
COM.ibm.db2.app.Lob 366
COM.ibm.db2.app.UDF 364
estilo de parámetros para rutinas externas 95
javaheapsz, parámetro de configuración 186
jdk11path, parámetro de configuración 186
Java *(continuación)*
métodos COM.ibm.db2.app.UDF 358
modelo de ejecución de las funciones de tabla 65
paquetes y clases
COM.ibm.db2.app 185
procedimientos almacenados 181
DB2DBG.ROUTINE_DEBUG, tabla de depuración 193
depuración 189
invocación del depurador 191
manejo de parámetros 56
para transformadores de almacén 187
preparación de la depuración 190
rutinas 181
DB2GENERAL 357
rendimiento 24
tipos de datos
BigDecimal 185
Blob 185
Double 185
Int 185
java.math.BigDecimal 185
Short 185
String 185
UDF (funciones definidas por el usuario) 358
áreas reutilizables 358
FENCED 358
NOT FENCED 358
sentencia CALL para archivos JAR 187
java.math.BigDecimal, tipo de datos de Java 185
javaheapsz, parámetro de configuración 186
jdk11path, parámetro de configuración 186
jerarquía 268, 270

L

LANGUAGE OLE, cláusula
CREATE FUNCTION, sentencia 195
lenguaje C/C++
rutinas 164
lenguaje COBOL
procedimientos almacenados 369
tipos de datos 371
LOB (objeto grande), tipos de datos
extracción de información de
Extracción de información de LOB con activadores 351
pase a rutinas 228
uso de 235
variables de referencia a archivo 242
localizadores de LOB
recuperación de un valor LOB con un localizador de lob 238
LONG VARCHAR, tipo de datos
COBOL 371
función de tabla de OLE DB 206
funciones definidas por el usuario (UDF)
C/C 171
Java 185
LONG VARCHAR, tipo de datos *(continuación)*
rutinas
Java (DB2GENERAL) 360
LONG VARCHAR FOR BIT DATA, tipo de datos
rutinas
Java (DB2GENERAL) 360
LONG VARGRAPHIC
parámetro de las UDF 171
LONG VARGRAPHIC, tipo de datos
COBOL 371
función de tabla de OLE DB 206
Java 185
rutinas
Java (DB2GENERAL) 360

M

manejadores
ejemplo 78
manejadores de condiciones
cláusula CONTINUE 81
ejemplo 78
procedimientos de SQL
declaración 78
descripción 78
sentencia RESIGNAL 81
sentencia SIGNAL 81
mensajes de error
visualización para procedimientos de SQL 77
métodos
alteración temporal del método 272
despacho dinámico de 272
para tipos estructurados
métodos de mutador 275
rutinas 3
visión general 18
métodos de observador 275
MODIFIES SQL DATA, cláusula
niveles de acceso a SQL en rutinas de SQL 69
rutinas externas 110
moneda
Definición de tipos diferenciados basados en la moneda 253
tablas que realizan un seguimiento de las ventas en diferentes monedas 255

N

niveles de aislamiento
rutinas 110
NO SQL, cláusula
rutinas externas 110
NOT FENCED, rutinas 27
NUMERIC, parámetro 171
NUMERIC, tipo de datos de SQL
COBOL 371
función de tabla de OLE DB 206
Java 185
rutinas
Java (DB2GENERAL) 360

O

Object Linking and Embedding (OLE) 194
objetos grandes (LOB)
  localizadores 236
OLE, rutinas de
  sintaxis para pasar argumentos 97
OLE DB
  funciones de tabla
    conexión de serie en la cláusula EXTERNAL NAME 203
    creación 203
    definidas por el usuario 201
    opción CONNECTSTRING 203
    utilizando el nombre de servidor 203
  nombres de fila completamente calificados 205
ONLY, cláusula
  restricción de los tipos devueltos mediante 296
OUTER, palabra clave
  devolución de atributos de subtipos 297

P

páginas de códigos
  rutinas, conversión 214
PARAMETER STYLE JAVA, rutinas 181
parámetros de configuración
  javaheapsz, parámetro de configuración 186
  jdk11path, parámetro de configuración 186
pase de LOB a rutinas 228
pase de tipos diferenciados a rutinas 227
pedido de publicaciones de DB2 394
PREPARE, sentencia
  SQL dinámico
    procedimientos de SQL 69
procedimiento
  devolución de conjuntos de resultados desde
    devolución de conjuntos de resultados desde procedimientos CLR 125
  llamada
    desde activadores 219
    desde aplicaciones y rutinas externas 217
    desde el Procesador de línea de mandatos (CLP) 222
    desde rutinas de SQL 219
  referencias (sintaxis de referencias de llamada) 215
procedimientos
  ejecución en el lenguaje común
    ejemplos de procedimientos CLR en C# 130
  manejo de parámetros 56
  recepción de conjuntos de resultados 53
    ejemplos de procedimientos CLR en C# 130
procedimientos *(continuación)*
  rutinas 3
procedimientos almacenados
  algoritmo de selección 216
  COBOL 369
  depuración
    centro de desarrollo 189
  devolución de conjuntos de resultados 47
  parámetros
    IN 46
    INOUT 46
    OUT 46
  referencias (sintaxis de referencias de llamada) 215
  selección 216
  visión general 13
procedimientos almacenados JDBC
  devolución de conjuntos de resultados 52
procedimientos de SQL
  CALL, sentencia 222
  devolución de conjuntos de resultados 49
  manejadores de condiciones
    declaración 78
  manejo de condiciones 78
  SQL dinámico 69
  variables SQLCODE y SQLSTATE 81
  visualización de mensajes de error 77
PROGRAM TYPE MAIN, cláusula
  procedimientos almacenados
    manejo de parámetros 56
PROGRAM TYPE SUB, cláusula
  procedimientos almacenados
    manejo de parámetros 56
programación, consideraciones
  rutinas, lenguajes soportados 21
publicaciones de DB2
  impresión de archivos PDF 393
publicaciones impresas, pedido 394
puntos de salvaguarda
  procedimientos 110

R

RAISE_ERROR, función escalar
  descripción 348
READS SQL DATA, cláusula
  rutinas externas 110
REAL, tipo de datos
  COBOL 371
  función de tabla de OLE DB 206
  funciones definidas por el usuario (UDF)
    C/C 171
  Java 185
  rutinas
    Java (DB2GENERAL) 360
recepción de conjuntos de resultados
  como rutina llamante 53
  en aplicaciones y rutinas JDBC 55
  en aplicaciones y rutinas SQLJ 54
REF USING, cláusula
  referencia a un tipo estructurado 286
referencias
  columnas 276, 291
  definición de relaciones 287
referencias con ámbito
  comparación con las referencias referenciales 290
REFERENCING, cláusula
  CREATE TRIGGER, sentencia 344
registro
  rutinas 35
rendimiento
  ajuste
    mediante rutinas 5
  rutinas 24
resolución de problemas
  guías de aprendizaje 398
  información en línea 398
restricciones
  interacción con activadores 336
  rutinas 32
RETURNS, cláusula
  CREATE FUNCTION, sentencia 171
REVOKE, sentencia
  emisión sobre jerarquías de tablas 276
rutinas
  anidadas 213
  automatización de OLE
    definición 195
  beneficios 5
  bibliotecas 30
  C/C++ 164
    tipos de datos de SQL soportados en 168
  clases 30
  CLR
    errores relacionados con 128
  conflictos de grabación 44
  conflictos de lectura 44
  cursores 110
  DB2GENERAL 357
    clases de Java 362
    COM.ibm.db2.app.Blob 367
    COM.ibm.db2.app.Clob 367
    COM.ibm.db2.app.Lob 366
  definición de la estructura del área reutilizable 61
  depuración 42
  emisión de sentencias CREATE 72
  escritura 36
  EXECUTE, privilegio 38
  externas
    actualización de rutinas de Java 187
    autorizaciones para 38
    ejecución en el lenguaje común 116, 117
    estilos de parámetros 95
    SQL en 110
    visión general 3
  funciones de tabla definidas por el usuario
    visión general 17
  invocación 209
    rutinas de 32 bits en un servidor de bases de datos de 64 bits 213
  Java 181

rutinas *(continuación)*
lenguajes de programación soportados 21
métodos 18
modificación 30
niveles de aislamiento 110
nombre 211
NOT FENCED
rendimiento 24
seguridad 27
opción WCHARTYPE del precompilador 179
páginas de códigos
conversión 214
pase de LOB a 228
pase de tipos diferenciados a 227
portabilidad entre plataformas de 32 bits y de 64 bits 61
procedimientos almacenados
visión general 13
recepción de conjuntos de resultados 53
registro 35
rendimiento 24
repetitivas 213
restricciones 32
rutinas de ejecución en el lenguaje común 116
devolución de conjuntos de resultados desde procedimientos CLR 125
ejemplos de funciones (UDF) CLR 152
Ejemplos de procedimientos CLR en C# 130
errores relacionados con 128
restricciones 126
tipos de datos de SQL soportados en 120
uso de la estructura Dbinfo 121
uso del área reutilizable 121
rutinas definidas por el usuario 10
seguridad 27
sintaxis para pasar argumentos 97
sobrecarga 211
SQL 3
THREADSAFE
rendimiento 24
seguridad 27
UDF escalares
visión general 15
variables del lenguaje principal de gráficos 179
vía de acceso a la función 211
rutinas DB2GENERAL 357
clases de Java 362
COM.ibm.db2.app.Blob 367
COM.ibm.db2.app.Clob 367
COM.ibm.db2.app.Lob 366
COM.ibm.db2.app.StoredProc 362
COM.ibm.db2.app.UDF 364
funciones definidas por el usuario 358, 364
procedimientos almacenados 362
rutinas de automatización de OLE
diseño 194
rutinas de SQL
ejemplos
Implementación de transformaciones de función utilizando rutinas de SQL 313
rutinas definidas por el usuario 10
rutinas externas 95

S

SCRATCHPAD, opción
conservación del estado 58
funciones definidas por el usuario (UDF) 58
rutinas de automatización de OLE 196
SELECT, sentencia
deshacer referencia, operadores 294
herencia de privilegios de las supertablas 276
referencias con ámbito 294
Sentencia CREATE METHOD
ejemplos 271
sentencias
CREATE FUNCTION 195
short, tipo de datos de Java 185
SIGNAL SQLSTATE
utilizado en los activadores 331
SMALLINT, tipo de datos
COBOL 371
función de tabla de OLE DB 206
funciones definidas por el usuario (UDF)
C/C 171
Java 185
rutinas
Java (DB2GENERAL) 360
sobrecarga
nombres de rutina 211
soporte de plataformas cruzadas
Invocación de rutinas de 32 bits en un servidor de bases de datos de 64 bits 213
SQL (Structured Query Language)
en rutinas
niveles de acceso al SQL en rutinas incorporadas al SQL 69
en rutinas externas 110
estilo de parámetros para rutinas externas 95
rendimiento de las rutinas 24
SQL dinámico
asignación de tipos 260
efectos de DYNAMICRULES 113
procedimientos de SQL 69
SQL estático
opciones de enlace de grupos de transformación para tipos estructurados 310
SQL-result, argumento
funciones de tabla 63
SQL-result-ind, argumento
funciones de tabla 63
SQLCODE
variables en procedimientos de SQL 81
sqldbchar, tipo de datos
en rutinas C/C++ 171
SQLJ (SQL incorporado para Java)
procedimientos almacenados
devolución de conjuntos de resultados 51
rutinas
contextos de conexión 189
SQLSTATE
emisión mediante sentencias SIGNAL y RESIGNAL 81
variables en procedimientos de SQL 81
SQLUDF, archivo de inclusión
rutinas en C/C++ 168
String, tipo de datos de Java 185
subtablas
creación 281
herencia de atributos 276
subtipos
devolución de atributos mediante OUTER 297
ejemplo 270
herencia 268
transformación, funciones de 322, 325
supertipos
columnas 270
en jerarquías de tipos estructurados 268

T

tabla de depuración
llenado 192
tablas
acceso
conflictos de lectura y grabación de rutinas 44
creación de tablas con columnas de tipo diferenciado 251
tablas con tipo
acceso a subtipos de la jerarquía de tipos 281
autorreferentes 288
columnas de identificador de objetos 276
control de los privilegios 276
creación 276
creación de subtablas 281
definición de relaciones 287, 288
definición del ámbito 276
descripción 281
determinación de la posición en la jerarquía 276
devolución de atributos de subtipos 297
eliminación
DROP TABLE, sentencia 279
implicaciones para los catálogos del sistema 279
opciones de columnas 276
restricciones 279
sustitución de tipos estructurados 280
tipos estructurados 279
visión general de 276

tablas de transición 344
THREADSAFE, rutinas 27
tiempo de activación
  tiempo de activación del activador 340
TIME, parámetro 171
TIME, tipo de datos
  COBOL 371
  función de tabla de OLE DB 206
  Java 185
  rutinas
    Java (DB2GENERAL) 360
TIMESTAMP, parámetro 171
TIMESTAMP, tipo de datos
  COBOL 371
  función de tabla de OLE DB 206
  Java 185
  rutinas
    Java (DB2GENERAL) 360
tipos
  diferenciados
    creación de tipos diferenciados 250
  estructurados
    creación de tipos estructurados 266
    tipo estructurado definido por el usuario 266
  tipos definidos por el usuario 247
tipos de columna
  creación
    COBOL 371
tipos de datos
  COBOL 371
  conversión
    entre DB2 y COBOL 371
    tipos de automatización de OLE 197
    transformación, funciones de 320
  diferenciados
    creación de tipos diferenciados 250
    manipulación 256
  estructurados
    acceso a atributos de tipo estructurado 304
    creación de tipos estructurados 266
    tipo estructurado definido por el usuario 266
  Java 185
  soportados
    COBOL, normas 371
tipos de datos COMP-1, en COBOL 371
tipos de datos COMP-3, en COBOL 371
tipos de datos COMP-5, en COBOL 371
tipos de datos de CLOB
  creación de un tipo diferenciado basado en un CLOB 254
tipos de datos de COBOL
  BLOB 371
  BLOB-FILE 371
  BLOB-LOCATOR 371
  cláusula PICTURE (PIC) 371
  cláusula USAGE 371
  CLOB 371
  CLOB-FILE 371
tipos de datos de COBOL *(continuación)*
  CLOB-LOCATOR 371
  COMP-1 371
  COMP-3 371
  COMP-5 371
  DBCLOB 371
  DBCLOB-FILE 371
  DBCLOB-LOCATOR 371
tipos de datos de LOB 240
  localizadores 236
  recuperación de valores LOB con un localizador de LOB 238
tipos de datos de objetos grandes (LOB) 240
  pase a rutinas 228
  utilización
    ejemplos de 235
  variables de referencia a archivo 242
tipos de datos de OLE DB
  conversión a tipos de datos de SQL 206
tipos de datos de SQL
  COBOL 371
  conversión a tipos de datos de OLE DB 206
  funciones definidas por el usuario (UDF) 171
  Java 185
  rutinas
    Java (DB2GENERAL) 360
  soportados en la automatización de OLE 197
tipos de datos definidos por el usuario
  creación
    con atributos de tipo estructurado 301
  tipos diferenciados
    tipificación estricta 249
tipos de datos diferenciados
  creación 250
    un tipo diferenciado basado en un CLOB 254
  creación de tablas con columnas de tipo diferenciado 251
tipos de referencia
  comparación 274
  comparación con las restricciones referenciales 287
  conversión del tipo de datos 274
  descripción 286
  operador para deshacer referencias 287
tipos de representación 286
tipos definidos por el usuario (UDT) 247
  eliminación de restricciones 252
  tipos diferenciados 247
    tipificación estricta 249
tipos diferenciados
  asignación de tipos de comparación 260
  comparación con otros tipos diferenciados 258, 261
  comparación con valores constantes 257, 259
tipos diferenciados *(continuación)*
  creación
    tipos diferenciados basados en la moneda 253
  creación de UDF con fuente para 262
  definidos por el usuario 247
  ejemplos de
    creación de una tabla para almacenar solicitudes de trabajo cumplimentadas 255
  manipulación 256
  pase a rutinas 227
  tipificación estricta de tipos diferenciados definidos por el usuario 249
  UNION, cláusulas 262
  uniones 262
  utilización de tipos diferenciados
    en tablas para almacenar diversas monedas 255
    para almacenar datos complejos 255
tipos dinámicos 280
tipos estáticos
  tipos estructurados basados en tipos estáticos 280
tipos estructurados
  actualización de atributos 303, 305
  almacenados en columnas de tabla 298
  almacenamiento 267
    como valor de una columna 267
  almacenamiento de instancias como filas 281
  atributos
    acceso a atributos de tipo estructurado 304
    actualización 305
    recuperación de atributos de tipo estructurado 304
  columnas de
    modificación de tablas con columnas de tipo estructurado 301
  columnas de referencia
    definición del ámbito 276
  como parámetros de rutinas
    Pase de parámetros de tipo estructurado a rutinas externas 314
  comparación de instancias con 296
  consulta
    selección de objetos de un tipo y no los subtipos 296
    utilización de la cláusula ONLY 296
    utilización de TYPE_NAME 306
  creación de tipos estructurados 266
  creación de una instancia de 274
  creación de una tabla con columnas de tipo estructurado 301
  creación de vistas con tipo 291
  declaración de variables del lenguaje principal 329
  definición de atributos 301
  definición del comportamiento
    ADD METHOD, cláusula 271

tipos estructurados *(continuación)*
definición del comportamiento *(continuación)*
Sentencia CREATE METHOD 271
DESCRIBE, sentencia 329
devolución de información sobre 306
eliminación de restricciones 252
funciones de constructor 274
herencia 268
herencia, control mediante la cláusula ONLY 276
identificadores de objetos
creación de restricciones 285
generación automática 283
inserción de instancias en columnas 300, 302
intercambio de valores con una rutina externa
utilización de funciones de transformación 321
intercambio de valores de tipo estructurado con programas del lenguaje principal 306
invocación de métodos 302
jerarquía 268, 270
ejemplos de 298
métodos de observador 275
métodos para 275
pase de instancias a aplicaciones cliente 316
pase de instancias a las rutinas externas 311
recuperación de atributos de subtipo 305
recuperación de instancias
como valores de atributos 275
como valores simples 303
recuperación del ID interno 294
recuperación del nombre de esquema 294
recuperación del nombre de tipo 294
referencia a objetos de fila 286
referencias
comparación con las restricciones referenciales 287
operador para deshacer referencias 287
requisitos de las funciones de transformación 321
subtipos
devolución de atributos mediante OUTER 297
tablas con tipo
acceso a subtipos 281
acceso a subtipos de la jerarquía de tipos 280
autorreferentes 288
columnas de identificador de objetos 276
control de los privilegios 276
creación 281
definición de relaciones 287
opciones de columnas 276
tipo estructurado definido por el usuario 266
tipos de los que no se pueden crear instancias 268
tipos estructurados *(continuación)*
tipos de los que se pueden crear instancias 268
tipos de representación 286
tipos dinámicos 280
tipos estáticos 280
transformación, funciones de 325
transformación, grupos de
denominación 307
especificación de grupos de transformación para rutinas externas 309
especificación de grupos de transformación para SQL dinámico 309
especificación de un grupo de transformación 308
transformaciones de funciones FROM SQL 311, 316
utilización
inserción en columnas de atributos de tipo estructurado 300
tipos raíz 268
transformación, funciones de 310
transformación, grupos de
especificación de un grupo de transformación para un tipo estructurado 308
transformaciones
funciones
asociación con tipos estructurados 306
enlace de subtipos 325
parámetros de subtipos 322
pase de objetos a las rutinas externas 311
pase de tipos estructurados a aplicaciones cliente 316
requisitos 321
grupos
denominación 307
rutinas externas 309
SQL dinámico 309
SQL estático 310
TREAT, expresión 305
TYPE, predicado
restricción de los tipos devueltos mediante 296
TYPE_ID, función
deshacer referencias de referencias 294
TYPE_NAME, función
deshacer referencias de referencias 294
TYPE_SCHEMA, función
deshacer referencias de referencias 294

U

UDF
funciones de tabla definidas por el usuario 63
UDF (funciones definidas por el usuario)
de tabla
FINAL CALL 63
NO FINAL CALL 63
UDF (funciones definidas por el usuario) *(continuación)*
de tabla, modelo de proceso 63
ejemplos de
Implementación de transformaciones de cliente mediante UDF externas 319
Implementación de transformaciones de cliente para enlazar desde un cliente mediante UDF externas 319
escalares, FINAL CALL 62
invocación 229
portabilidad del área reutilizable entre plataformas de 32 bits y de 64 bits 61
unión
sobre columnas con tipo diferenciado 262
uniones
tipos diferenciados 262

V

VARCHAR, tipo de datos
COBOL 371
función de tabla de OLE DB 206
Java 185
rutinas, Java (DB2GENERAL) 360
VARCHAR FOR BIT DATA, tipo de datos
funciones definidas por el usuario (UDF), C/C++ 171
rutinas, Java (DB2GENERAL) 360
VARGRAPHIC, tipo de datos
COBOL 371
función de tabla de OLE DB 206
funciones definidas por el usuario (UDF), C/C++ 171
Java 185
rutinas, Java (DB2GENERAL) 360
variables del lenguaje principal
declaración
tipos estructurados 329
tipos de datos de COBOL 371
variables del lenguaje principal de gráficos
rutinas 179
vinculación
rutinas 38
vista
vista con tipo 290
vistas
eliminación 293
eliminación, implicaciones para los catálogos del sistema 293
restricciones 293
tipos estructurados 293
vistas con tipo 290
asignación de un ámbito a las columnas de referencia 291
creación
sobre subtipos 291
sobre tipos raíz 291
cuerpo 291

W

wchart, tipo de datos 171
WCHARTYPE NOCONVERT, opción del precompilador 179
WITH OPTIONS, cláusula
  definición de opciones de columnas 276
  definición del ámbito de columnas de referencia 276

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

SC10-3724-01

Spine information:

IBM® DB2 Universal Database™ Aplicaciones de servidor de programación Versión 8.2