IBM DB2 10.1
para Linux, UNIX y Windows

Desarrollo de aplicaciones Java

Actualizado en enero de 2013

SC11-8065-01

IBM DB2 10.1
para Linux, UNIX y Windows

Desarrollo de aplicaciones Java

Actualizado en enero de 2013

SC11-8065-01

Nota

Antes de utilizar esta información y el producto al que se aplica, lea la información general que encontrará en el apartado Apéndice B, "Avisos", en la página 645.

Nota de edición

Este manual es la traducción del original en inglés *IBM DB2 10.1 for Linux, UNIX, and Windows Developing Java Applications Updated January, 2013* (SC27-3875-01).

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

- Para solicitar publicaciones en línea, vaya al Centro de publicaciones de IBM en http://www.ibm.com/shop/publications/order
- Para encontrar el representante local de IBM, vaya a IBM Directory of Worldwide Contacts en http://www.ibm.com/planetwide/

Para realizar pedidos de publicaciones de DB2 desde DB2 Marketing and Sales, en los EE.UU. o en Canadá, llame al 1-800-IBM-4YOU (426-4968).

Contenido

Acerca de este manual

Este manual describe el soporte de DB2 para Linux, UNIX y Windows para Java. Este soporte le permite acceder a bases de datos relacionales desde programas de aplicación Java.

A quién va dirigido este manual

Este manual va dirigido a los siguientes usuarios:

- Desarrolladores de aplicaciones de DB2 para Linux, UNIX y Windows que estén familiarizados con Structured Query Language (SQL) y que conozcan el lenguaje de programación Java.
- Programadores del sistema de DB2 para Linux, UNIX y Windows que vayan a instalar el soporte de JDBC y SQLJ.

Capítulo 1. Desarrollo de aplicaciones Java para servidores de datos de IBM

Los sistemas de base de datos DB2 e IBM® Informix proporcionan soporte de controlador para aplicaciones cliente y applets escritos en Java.

Puede acceder a datos en sistemas de base de datos DB2 e IBM Informix utilizando JDBC, SQL o pureQuery.

JDBC

JDBC es una interfaz de programación de aplicaciones (API) que las aplicaciones Java utilizan para acceder a las bases de datos relacionales. El soporte de IBM Data Server para JDBC le permite grabar aplicaciones Java que acceden a datos locales de DB2 o IBM Informix o a datos relacionales remotos de un servidor que admite DRDA.

SQLJ

SQLJ proporciona soporte para SQL estático incorporado en aplicaciones Java. IBM, Oracle y Tandem desarrollaron inicialmente SQLJ, para complementar al modelo JDBC de SQL dinámico con un modelo de SQL estático.

Para las conexiones a DB2, en general, las aplicaciones Java utilizan JDBC para el SQL dinámico y SQLJ para el SQL estático.

Para las conexiones con IBM Informix, las sentencias de SQL contenidas en aplicaciones JDBC o SQLJ se ejecutan dinámicamente.

Debido a que SQLJ puede interactuar con JDBC, un programa de aplicación puede utilizar JDBC y SQLJ dentro de la misma unidad de trabajo.

pureQuery

pureQuery es una plataforma de acceso a datos de alto rendimiento que facilita el desarrollo, la optimización, la protección y la gestión del acceso a los datos. Se compone de:

- Interfaces de programación de aplicaciones creadas para facilitar el uso y simplificar el uso de las recomendaciones.
- Las herramientas de desarrollo, que se entregan en IBM InfoSphere Optim Development, para el desarrollo de Java y SQL.
- Un tiempo de ejecución, que se proporciona en IBM InfoSphere Optim pureQuery Runtime, para optimizar y proteger el acceso a la base de datos y simplificar las tareas de gestión.

Con pureQuery, puede escribir aplicaciones Java que tratan los datos relacionales como objetos, tanto si esos datos se encuentran en bases de datos como en objetos JDBC DataSource. Sus aplicaciones también pueden tratar los objetos almacenados en colecciones Java en memoria como si dichos objetos fueran datos relacionales. Para consultar o actualizar los datos relacionales o los objetos Java, utilice SQL.

Para obtener más información sobre pureQuery, consulte el Centro de información de Integrated Data Management.

Controladores soportados para JDBC y SQLJ

El producto DB2 incluye soporte para dos tipos de arquitectura del controlador JDBC.

De acuerdo con la especificación JDBC, existen cuatro tipos de arquitecturas de controlador JDBC:

Tipo 1
Son controladores que implementan la API de JDBC como una correlación con otra API de acceso a datos, como por ejemplo Open Database Connectivity (ODBC). Los controladores de este tipo generalmente dependen de una biblioteca nativa, lo cual limita su portabilidad. El sistema de bases de datos DB2 no proporciona un controlador de tipo 1.

Tipo 2
Son controladores que están escritos parcialmente en el lenguaje de programación Java y parcialmente en código nativo. Estos controladores utilizan una biblioteca cliente nativa que es específica de la fuente de datos a la que se conectan. Debido al código nativo, la portabilidad de estos controladores es limitada.

Tipo 3
Son controladores que utilizan un cliente de Java puro y se comunican con un servidor de datos utilizando un protocolo independiente del servidor de datos. A continuación, el servidor de datos transmite las peticiones del cliente a la fuente de datos. El sistema de bases de datos DB2 no proporciona un controlador de tipo 3.

Tipo 4
Estos controladores son Java puro e implementan el protocolo de red de una fuente de datos determinada. El cliente se conecta directamente con la fuente de datos.

DB2 Database para Linux, UNIX y Windows es compatible con el controlador siguiente:

Nombre de controlador	Empaquetado como	Tipo de controlador
IBM Data Server Driver para JDBC y SQLJ	• db2jcc.jar y sqlj.zip para JDBC 3.0, soporte • db2jcc4.jar y sqlj4.zip para soporte de determinadas funciones de JDBC 4.0 o posterior	Tipo 2 y Tipo 4

IBM Data Server Driver para JDBC y SQLJ (tipo 2 y tipo 4)

IBM Data Server Driver para JDBC y SQLJ es un controlador individual que incluye comportamiento propio de los tipos 2 y 4 de JDBC. Cuando una aplicación carga IBM Data Server Driver para JDBC y SQLJ, se carga una instancia de controlador para las implementaciones de tipo 2 y tipo 4. La aplicación puede establecer conexiones de tipo 2 y tipo 4 utilizando esta instancia de controlador. Las conexiones de tipo 2 y tipo 4 se pueden establecer simultáneamente. Al comportamiento de IBM Data Server Driver para JDBC y SQLJ de tipo 2 se le hace

referencia como *IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2.* Al comportamiento de IBM Data Server Driver para JDBC y SQLJ de tipo 4 se le hace referencia como *IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4.*

Existen dos versiones disponibles de IBM Data Server Driver para JDBC y SQLJ. El controlador IBM Data Server Driver para JDBC y SQLJ versión 3.5*x* es compatible con JDBC 3.0. IBM Data Server Driver para JDBC y SQLJ versión 4.*x* es compatible con JDBC 4.0 o posterior.

IBM Data Server Driver para JDBC y SQLJ soporta estas funciones JDBC y SQLJ:

- La Versión 3.5*x* da soporte a todos los métodos que se describen en las especificaciones de JDBC 3.0.
- Versión 4.*x* da soporte a todos los métodos que se describen en las especificaciones de JDBC 4.0 o posterior.
- Interfaces de programación de aplicaciones SQLJ, tal como definen las normas de SQLJ, para lograr un acceso simplificado a los datos desde aplicaciones Java.
- Conexiones que están habilitadas para la agrupación de conexiones. WebSphere Application Server u otro servidor de aplicaciones realiza la agrupación de conexiones.
- Las conexiones a un servidor de datos desde funciones definidas por el usuario y procedimientos almacenados de Java utilizan sólo IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2. Las aplicaciones que invocan funciones definidas por el usuario y procedimientos almacenados pueden utilizar IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 o IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 para conectarse a un servidor de datos.

  IBM Data Server Driver para JDBC y SQLJ es el controlador por omisión de las rutinas Java.
- Soporte para la gestión de transacciones distribuidas. Este soporte implementa las especificaciones Java 2 Platform, Enterprise Edition (J2EE) Java Transaction Service (JTS) y Java Transaction API (JTA), que se ajustan al estándar de X/Open para transacciones distribuidas (vea la publicación *Distributed Transaction Processing: The XA Specification* en el sitio Web http://www.opengroup.org).

Compatibilidad entre versiones del controlador JDBC y la base de datos

La compatibilidad de una versión determinada de IBM Data Server Driver para JDBC y SQLJ con una versión de la base de datos depende del tipo de conectividad de controlador que esté utilizando y el tipo de fuente de datos al que se conecte.

Compatibilidad para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4

IBM Data Server Driver para JDBC y SQLJ es siempre compatible con bases de datos DB2 pertenecientes a un nivel de release anterior. Por ejemplo, se da soporte a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 desde IBM Data Server Driver para JDBC y SQLJ versión 3.61, que se proporciona con DB2 Database para Linux, UNIX y Windows Versión 9.7 Fixpack 3, con una base de datos DB2 Database para Linux, UNIX y Windows Versión 8.

IBM Data Server Driver para JDBC y SQLJ es compatible con la versión siguiente de una base de datos DB2 si las aplicaciones bajo las que se ejecuta el controlador no utilizan funciones nuevas. Por ejemplo, está permitido IBM Data Server Driver

para JDBC y SQLJ con conectividad de tipo 4 de IBM Data Server Driver para JDBC y SQLJ versión 2.*x*, que se proporciona con DB2 para z/OS Versión 8, con una base de datos DB2 para z/OS Versión 9.1 si las aplicaciones bajo las que se ejecuta el controlador no contienen funciones de DB2 para z/OS Versión 9.1.

IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 con IBM Informix solamente se soporta para IBM Informix Versión 11 y posterior.

Compatibilidad para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2

En general, IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 está pensado para conexiones con el sistema de bases de datos local, utilizando la versión de controlador que se proporciona con esa versión de base de datos. Por ejemplo, la Versión 3.6*x* de IBM Data Server Driver para JDBC y SQLJ se proporciona con DB2 Database para Linux, UNIX y Windows Versión 9.5 y Versión 9.7 y DB2 para z/OS Versión 8 y versiones posteriores.

Pero para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 con una base de datos local DB2 Database para Linux, UNIX y Windows, la versión de la base de datos puede ser una versión anterior o posterior que la versión de DB2 Database para Linux, UNIX y Windows que se proporciona con el controlador. Para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 con un subsistema local DB2 para z/OS, la versión del subsistema puede ser una versión posterior que la versión de DB2 para z/OS que se proporciona con el controlador.

Si la versión de la base de datos a la que se conectan las aplicaciones es posterior a la versión de la base de datos que se proporciona con el controlador, las aplicaciones no pueden utilizar funciones de la versión de base de datos posterior.

Niveles de DB2 para Linux, UNIX y Windows y de IBM Data Server Driver para JDBC y SQLJ

Cada versión de DB2 para Linux, UNIX y Windows se entrega con una versión JDBC 3 y una versión JDBC 4 de IBM Data Server Driver para JDBC y SQLJ.

En la tabla siguiente se enumeran las versiones de DB2 para Linux, UNIX y Windows y las versiones de IBM Data Server Driver para JDBC y SQLJ correspondientes. Puede utilizar esta información para determinar el nivel de DB2 para Linux, UNIX y Windows o de DB2 Connect que está asociado con la instancia de IBM Data Server Driver para JDBC y SQLJ en la que se ejecuta un programa cliente.

Tabla 1. Niveles de fixpack de DB2 Database para Linux, UNIX y Windows y versiones de IBM Data Server Driver para JDBC y SQLJ

Versión y nivel de fixpack de DB2	Versión de IBM Data Server Driver para JDBC y SQLJ[1]
DB2 Versión 10.1 Fixpack 2	3.65.*xx*, 4.15.*xx*
DB2 Versión 10.1 Fixpack 1	3.64.*xx*, 4.14.*xx*
DB2 Versión 10.1	3.63.*xx*, 4.13.*xx*
DB2 Versión 9.7 Fixpack 6	3.64.*xx*, 4.14.*xx*
DB2 Versión 9.7 Fixpack 5	3.63.*xx*, 4.13.*xx*
DB2 Versión 9.7 Fixpack 4	3.62.*xx*, 4.12.*xx*

Tabla 1. Niveles de fixpack de DB2 Database para Linux, UNIX y Windows y versiones de IBM Data Server Driver para JDBC y SQLJ (continuación)

Versión y nivel de fixpack de DB2	Versión de IBM Data Server Driver para JDBC y SQLJ[1]
DB2 Versión 9.7 Fixpack 2	3.59.*xx*, 4.9.*xx*
DB2 Versión 9.7 Fixpack 1	3.58.*xx*, 4.8.*xx*
DB2 Versión 9.7	3.57.*xx*, 4.7.*xx*
DB2 Versión 9.5 Fixpack 7	3.61.*xx*, 4.8.*xx*
DB2 Versión 9.5 Fixpack 6	3.58.*xx*, 4.8.*xx*
DB2 Versión 9.5 Fixpack 5	3.57.*xx*, 4.7.*xx*
DB2 Versión 9.5 Fixpack 3 y Fixpack 4	3.53.*xx*, 4.3.*xx*
DB2 Versión 9.5 Fixpack 2	3.52.*xx*, 4.2.*xx*
DB2 Versión 9.5 Fixpack 1	3.51.*xx*, 4.1.*xx*
DB2 Versión 9.5	3.50.*xx*, 4.0.*xx*
DB2 Versión 9.1 Fixpack 5 y posterior	3.7.*xx*
DB2 Versión 9.1 Fixpack 4	3.6.*xx*
DB2 Versión 9.1 Fixpack 3	3.4.*xx*
DB2 Versión 9.1 Fixpack 2	3.3.*xx*
DB2 Versión 9.1 Fixpack 1	3.2.*xx*
DB2 Versión 9.1	3.1.*xx*

Nota:

1. Todas las versiones de los controladores están en el formato *n.m.xx*. *n.m* no cambia dentro del mismo nivel de GA o nivel de fixpack. *xx* cambia cuando se incorpora una versión nueva de IBM Data Server Driver para JDBC y SQLJ mediante un arreglo APAR.

Encontrará información más detallada sobre las versiones de IBM Data Server Driver para JDBC y SQLJ y DB2 Database para Linux, UNIX y Windows en el URL siguiente:

`http://www.ibm.com/support/docview.wss?&uid=swg21363866`

Capítulo 2. Instalación de IBM Data Server Driver para JDBC y SQLJ

Después de instalar IBM Data Server Driver para JDBC y SQLJ, puede preparar y ejecutar aplicaciones JDBC o SQLJ.

Antes de empezar

Antes de instalar IBM Data Server Driver para JDBC y SQLJ, necesita el software siguiente.

- Un SDK para Java, 1.4.2 o posterior.

  Para todos los productos DB2 excepto IBM Data Server Runtime Client e IBM Data Server Driver Package, el proceso de instalación de DB2 Database para Linux, UNIX y Windows instala automáticamente el SDK para Java, Versión 5.

  Si desea utilizar funciones de JDBC 4.0, necesita instalar un SDK para Java, Versión 6 o posterior.

  Si desea utilizar funciones de JDBC 4.1, necesita instalar un SDK para Java, Versión 7 o posterior.

  Si piensa ejecutar aplicaciones JDBC o SQLJ en el sistema, pero no prepararlas, necesita solamente un entorno de ejecución Java.

  Importante: El soporte para el SDK para Java 1.4.2 ha quedado en desuso para las rutinas Java y es probable que deje de mantenerse en un futuro release.
- Soporte de hebras nativas de JVM

  Todas las JVM que ejecutan aplicaciones Java que acceden a bases de datos DB2 deben incluir soporte de hebras nativas. Puede especificar hebras nativas como soporte de hebras por omisión para algunas JVM asignando el valor "native" a la variable de entorno THREADS_FLAG. Consulte la documentación del entorno Java para conocer las instrucciones sobre cómo hacer que las hebras nativas sean las hebras por omisión en su sistema.
- Soporte de Unicode para servidores System i

  Si hay algún programa SQLJ o JDBC que utilice IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 para conectarse a un servidor DB2 para i, el sistema operativo System i deberá dar soporte al esquema de codificación UTF-8 de Unicode. La tabla siguiente lista los PTF de System i que son necesarios para dar soporte a UTF-8 de Unicode:

Tabla 2. PTF de System i para el soporte de UTF-8 de Unicode

Versión de System i	Números de PTF
V5R3 o posterior	Ninguno (el soporte está incluido)

- Soporte de Java a clientes y servidores HP-UX

  Servidores HP-UX: IBM Data Server Driver para JDBC y SQLJ no da soporte a las bases de datos que hacen uso del juego de caracteres por omisión de HP-UX, Roman8. Por tanto, cuando cree una base de datos en un servidor HP-UX al que piense acceder mediante IBM Data Server Driver para JDBC y SQLJ, es necesario que cree la base de datos con un juego de caracteres diferente.

Clientes y servidores de HP-UX: el entorno de Java de un sistema HP-UX requiere una configuración especial para poder ejecutar procedimientos almacenados en IBM Data Server Driver para JDBC y SQLJ.

Acerca de esta tarea

Restricción: Si instala IBM Data Server Driver para JDBC y SQLJ en un sistema operativo Windows de 64 bits, no podrá utilizar IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 para conectar con una instancia de DB2 Database para Linux, UNIX y Windows desde una aplicación Java de 32 bits.

Siga estos pasos para instalar el IBM Data Server Driver para JDBC y SQLJ.

Procedimiento

1. Durante el proceso de instalación de DB2 Database para Linux, UNIX y Windows, seleccione `Soporte de Java` en UNIX o Linux, o `Soporte de JDBC` en Windows. Son las opciones por omisión. Si ya ha instalado DB2 Database para Linux, UNIX y Windows sin el soporte de JDBC, puede ejecutar el proceso de instalación en modalidad `Personalizada` para añadir el soporte de JDBC.

   La selección de `Soporte Java` o `Soporte JDBC` hace que el proceso de instalación realice estas acciones:

   a. Instala los archivos de clase de IBM Data Server Driver para JDBC y SQLJ.

      Los archivos se colocan en el directorio sqllib\java en los sistemas Windows, o en el directorio sqllib/java en los sistemas UNIX o Linux.

      Los nombres de archivo son:

      db2jcc.jar o db2jcc4.jar

      Incluya db2jcc.jar en la variable CLASSPATH si piensa utilizar la versión de IBM Data Server Driver para JDBC y SQLJ que incluye solamente **JDBC 3.0 o funciones anteriores.**

      Incluya db2jcc4.jar en la variable CLASSPATH si piensa utilizar la versión de IBM Data Server Driver para JDBC y SQLJ que incluye **JDBC 4.0 o funciones anteriores, y JDBC 3.0 o funciones anteriores.**

      sqlj.zip o sqlj4.zip

      Incluya sqlj.zip en la variable CLASSPATH si piensa preparar aplicaciones de SQLJ que sólo incluyan **JDBC 3.0 o funciones anteriores.**

      Incluya sqlj4.zip en la variable CLASSPATH si piensa preparar aplicaciones SQLJ que incluyan **JDBC 4.0 o funciones anteriores, y JDBC 3.0 o funciones anteriores.**

   b. Modifica la variable CLASSPATH para que incluya los archivos de clase de IBM Data Server Driver para JDBC y SQLJ.

      Importante: Este paso se ejecuta automáticamente solamente para los archivos db2jcc.jar y sqlj.zip. Si está utilizando el archivo db2jcc4.jar o el archivo sqlj4.zip, debe especificar la variable CLASSPATH manualmente. Cambie db2jcc.jar por db2jcc4.jar o sqlj.zip por sqlj4.zip en la variable CLASSPATH.

      Tendrá que realizar también este cambio en todas las ventanas de línea de mandatos de DB2 que abra.

Importante: Incluya db2jcc.jar o db2jcc4.jar en CLASSPATH. No incluya ambos archivos.

Importante: Incluya sqlj.zip o sqlj4.zip en la variable CLASSPATH. No incluya ambos archivos. No incluya db2jcc.jar con sqlj4.zip, ni db2jcc4.jar con sqlj.zip.

c. Si existen archivos de licencia de cliente de IBM Data Server Driver para JDBC y SQLJ, el proceso de instalación los instala y modifica la variable CLASSPATH para incluirlos.

   Los archivos se colocan en el directorio sqllib\java en los sistemas Windows, o en el directorio sqllib/java en los sistemas UNIX o Linux. Los nombres de archivo son:

Tabla 3. Archivos de licencia de IBM Data Server Driver para JDBC y SQLJ

Archivo de licencia	Servidor al que el archivo de licencia permite una conexión	Producto donde se incluye el archivo de licencia
db2jcc_license_cisuz.jar	DB2 para z/OS DB2 para i	Todos los productos de DB2 Connect

Los archivos de licencia de cliente no son necesarios para las conexiones con bases de datos DB2 Database para Linux, UNIX y Windows, Cloudscape o IBM Informix realizadas desde IBM Data Server Driver para JDBC y SQLJ versión 3.50 o posterior. Los archivos de licencia de cliente no son necesarios para las conexiones directas con DB2 para z/OS si se lleva a cabo la activación de licencias del servidor DB2 Connect Unlimited Edition for System z.

Recomendación: Si se conecta a los servidores de datos DB2 para z/OS directamente, en lugar de hacerlo a través de una pasarela DB2 Connect, y utiliza DB2 Connect Unlimited Edition for System z 9.7 Fixpack 3 o posterior, debe activar la clave de licencia del servidor para DB2 Connect Unlimited Edition for System z. Debe activar la clave de licencia en cada subsistema o grupo de compartimiento de datos de DB2 para z/OS al que se vaya a conectar. Consulte la información sobre DB2 Connect para ver los detalles sobre la activación de licencias de servidor.

d. Instala las bibliotecas nativas de IBM Data Server Driver para JDBC y SQLJ para poder utilizar IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2.

   Los archivos se colocan en el directorio sqllib\bin en los sistemas Windows, o en el directorio sqllib/lib en los sistemas UNIX o Linux.

   Los nombres de archivo son:

   libdb2jcct2.so
   Para AIX, HP-UX en IPF, Linux y Solaris

   db2jcct2.dll
   Para Windows

Como alternativa a instalar los archivos de clases de IBM Data Server Driver para JDBC y SQLJ durante la instalación, puede descargar los archivos de clases y seguir los pasos anteriores para configurar el controlador. No puede descargar las bibliotecas nativas de IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2. Para descargar los archivos de clases de IBM Data Server Driver para JDBC y SQLJ, siga estos pasos:

•

- Vaya a http://www.ibm.com/software/data/support/data-server-clients/download.html.
- En Descargas y arreglos, seleccione View IBM Data Server Client Packages.
- En la ventana Refinar mi lista de arreglos, seleccione Mostrarme más opciones.
- En la página Fix Central, seleccione Information Management en el campo Grupo de productos, IBM Data Server Client Packages en el campo Producto, la versión más reciente en el campo Versión instalada y Todo en el campo Plataforma.
- En la página Identificar arreglos, escriba "Data Server Driver for JDBC" en el campo Texto.
- En la página Seleccionar arreglos, seleccione la versión más reciente de IBM Data Server Driver para JDBC y SQLJ.
- En la página Opciones de descarga, seleccione las opciones que se adapten a sus necesidades.

• Extraiga el archivo zip en un directorio vacío.

El archivo zip contiene los archivos siguientes:

- **db2jcc.jar**
- **db2jcc4.jar**
- **sqlj.zip**
- **sqlj4.zip**

• Copie los archivos en las ubicaciones que se especifican en el paso 1a en la página 8 anterior.

Tras haber descargado los archivos de clases de IBM Data Server Driver para JDBC y SQLJ, tendrá que seguir todo el procedimiento que se describe en este tema para instalar el controlador.

2. Personalice las propiedades de configuración del controlador si cualquiera de los valores por omisión no son adecuados.
3. Configure TCP/IP.

   Los servidores se deben configurar para la comunicación TCP/IP en estos casos:

   • Aplicaciones JDBC o SQLJ que utilizan IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4.
   • Aplicaciones JDBC o SQLJ que utilizan IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2, y especifique *servidor* y *puerto* en el URL de conexión.

   Asegúrese de que el TCP/IP listener se está ejecutando. Para activar el TCP/IP listener:

   a. Defina con el valor TCPIP la variable de entorno DB2COMM:

      ```
      db2set DB2COMM=TCPIP
      ```

   b. Actualice el archivo de configuración del gestor de bases de datos con el nombre del servicio de TCP/IP que se haya especificado en el archivo services:

      ```
      db2 update dbm cfg using SVCENAME nombre_servicio_TCP/IP
      ```

      El número de puerto utilizado para los applets y los programas de SQLJ necesita ser el mismo que el número de SVCENAME de TCP/IP utilizado en el archivo de configuración del gestor de bases de datos.

c. Ejecute los mandatos db2stop y db2start para que el valor del nombre de servicio entre en vigor.

4. En los servidores DB2 Database para Linux, UNIX y Windows en los que piense ejecutar procedimientos almacenados Java o funciones definidas por el usuario, actualice la configuración del gestor de bases de datos para incluir la vía de acceso donde reside el SDK para Java.

   Puede hacerlo entrando mandatos similares a éstos en la línea de mandatos del servidor:

   - *Para los sistemas de bases de datos en UNIX o Linux:*

     ```
     db2 update dbm cfg using JDK_PATH /home/db2inst/jdk15
     ```

     /home/db2inst/jdk15 es la vía de acceso donde está instalado el SDK para Java.

   - *Para los sistemas de bases de datos en Windows:*

     ```
     db2 update dbm cfg using JDK_PATH c:\Archivos de programa\jdk15
     ```

     c:\Archivos de programa\jdk15 es la vía de acceso donde está instalado el SDK para Java.

   Para verificar el valor correcto para el campo JDK_PATH en la configuración del gestor de bases de datos DB2, emita el mandato siguiente en el servidor de bases de datos:

   ```
   db2 get dbm cfg
   ```

   Puede ser conveniente redirigir la salida del mandato hacia un archivo para facilitar su legibilidad. El campo JDK_PATH aparece cerca del comienzo de los datos de salida del mandato.

5. Si piensa invocar procedimientos SQL que residen en servidores DB2 Database para Linux, UNIX y Windows desde programas Java, y el formato de fecha y hora que está asociado al código de territorio de los servidores de bases de datos **no** corresponde al formato utilizado en Estados Unidos, siga estos pasos:

   a. Defina la variable de registro DB2_SQLROUTINE_PREPOPTS en los servidores de bases de datos para indicar que el formato de la fecha y la hora por omisión es ISO:

      ```
      db2set DB2_SQLROUTINE_PREPOPTS="DATETIME ISO"
      ```

   b. Vuelva a definir los procedimientos SQL existentes que tenga previsto invocar desde programas Java.

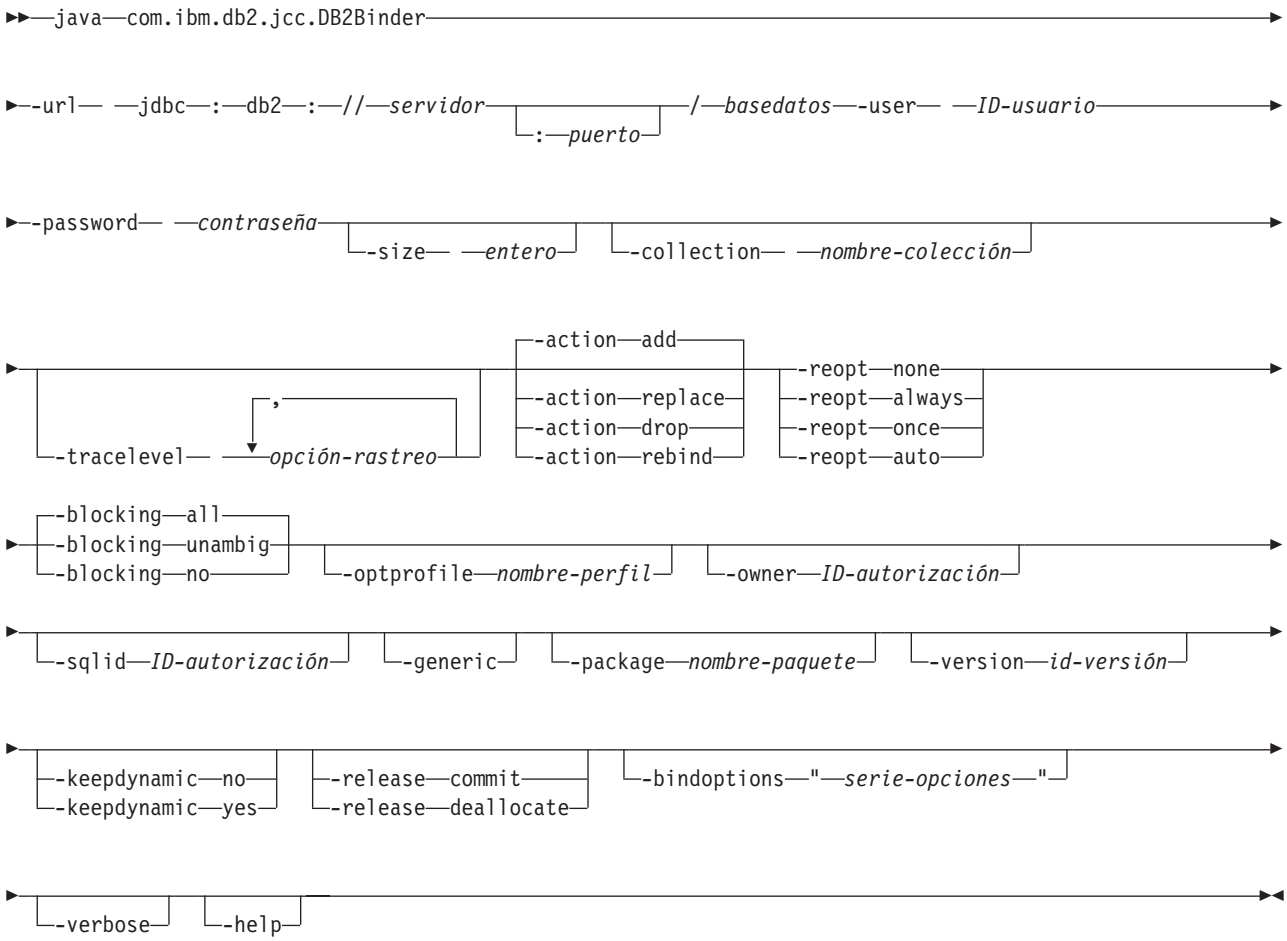
   Estos pasos son necesarios para garantizar que la aplicación de llamada recibe correctamente los valores de fecha y hora.

6. Si tiene intención de acceder a servidores de bases de datos DB2 para z/OS con sus aplicaciones Java, siga las instrucciones de "Instalación especial para acceder a servidores DB2 para z/OS desde programas Java" en *Desarrollo de aplicaciones Java*.

Programa de utilidad DB2Binder

El programa de utilidad DB2Binder vincula los paquetes de DB2 utilizados en el servidor de datos por IBM Data Server Driver para JDBC y SQLJ, y otorga autorización EXECUTE sobre los paquetes a PUBLIC. Opcionalmente, el programa de utilidad DB2Binder puede revincular paquetes DB2 que no forman parte de IBM Data Server Driver para JDBC y SQLJ.

Sintaxis de DB2Binder



Descripciones de la opción DB2Binder

-url
Especifica la fuente de datos en la que se deben vincular los paquetes de IBM Data Server Driver para JDBC y SQLJ. Las partes variables del valor `-url` son:

servidor
Nombre de dominio o dirección IP del sistema operativo donde reside el servidor de datos.

puerto
El número de puerto del servidor TCP/IP que está asignado al servidor de datos. El valor por omisión es 446.

basedatos
Nombre de ubicación del servidor de datos, tal como está definido en la tabla de catálogo SYSIBM.LOCATIONS.

-user
Especifica el ID de usuario utilizado para vincular los paquetes. Este usuario debe tener autorización BIND sobre los paquetes.

-action
Especifica la acción que debe realizarse en los paquetes.

add Indica que puede crearse un paquete sólo si no existe ya. Add es el valor por omisión.

replace
Indica que puede crearse un paquete aunque ya exista otro paquete con el mismo nombre. El paquete nuevo sustituirá al anterior.

rebind
Indica que el paquete existente se debe volver a vincular. Esta opción no es aplicable a los paquetes de IBM Data Server Driver para JDBC y SQLJ. Si se especifica -action rebind, se debe especificar también -generic.

drop Indica que se deben descartar los paquetes:

- Para los paquetes IBM Data Server Driver para JDBC y SQLJ, la opción -action drop indica que se deben descartar algunos o todos los paquetes de IBM Data Server Driver para JDBC y SQLJ. El número de paquetes depende del parámetro -size.
- Para los paquetes de usuario, la opción -action drop indica que se debe eliminar el paquete especificado.

-action drop solamente es aplicable si el servidor de datos de destino es DB2 para z/OS.

-size
Controla el número de objetos Statement, PreparedStatement o CallableStatement que pueden estar abiertos a la vez, o el número de paquetes de IBM Data Server Driver para JDBC y SQLJ que se descartan.

El significado del parámetro -size depende del parámetro -action:

- Si el valor de -action es add o replace, el valor de -size es un número entero que determina el número de paquetes de DB2 que son vinculados por IBM Data Server Driver para JDBC y SQLJ. Si el valor de -size es *número-entero*, el número total de paquetes es:

  ```
  número-de-niveles-de-aislamiento*
  número-de-valores-de-capacidad-de-retención*
  número-entero+
  número-de-paquetes-para-SQL-estático
  = 4*2*número-entero+1
  ```

  El valor por omisión de -size para -action add o -action replace es 3.

  En la mayoría de los casos, el valor por omisión 3 es apropiado. Si sus aplicaciones emiten SQLException con el código de SQL -805, compruebe que las aplicaciones cierran todos los recursos no utilizados. Si lo hacen, aumente el valor de -size.

  Si el valor de -action es replace, y el valor de -size da lugar a un número menor de paquetes que los ya existentes, no se descarta ningún paquete.
- Si el valor de -action es drop, el valor de -size es el número de paquetes que se eliminan. Si -size no está especificado, se descartan todos los paquetes de IBM Data Server Driver para JDBC y SQLJ.
- Si el valor de -action es rebind, -size no se tiene en cuenta.

-collection
Especifica el ID de colección para paquetes de IBM Data Server Driver para JDBC y SQLJ o paquetes de usuario. El valor por omisión es NULLID. DB2Binder convierte este valor a mayúsculas.

Puede crear varias instancias de los paquetes de IBM Data Server Driver para JDBC y SQLJ en un mismo servidor de datos ejecutando

`com.ibm.db2.jcc.DB2Binder` varias veces, especificando un valor diferente para -collection cada vez. Durante la ejecución, seleccione una instancia de IBM Data Server Driver para JDBC y SQLJ asignando a la propiedad `currentPackageSet` un valor que coincida con valor de -collection.

-tracelevel
Especifica qué se debe rastrear mientras se ejecuta DB2Binder.

-reopt
Especifica si los servidores de datos determinan las vías de acceso durante la ejecución. Esta opción no se envía al servidor de datos si no se especifica. En este caso, el servidor de datos determina el comportamiento de reoptimización.

-reopt se aplica a conexiones con DB2 para z/OS Versión 8 o posterior, o bien DB2 Database para Linux, UNIX y Windows Versión 9.1 o posterior.

none Especifica que las vías de acceso no se determinan en tiempo de ejecución.

always
Especifica que las vías de acceso se determinan cada vez que se ejecuta una sentencia.

once Especifica que DB2 determina y almacena la vía de acceso para una sentencia dinámica una sola vez durante la ejecución. DB2 utiliza esta vía de acceso hasta que se invalida la sentencia preparada. o hasta que la sentencia se elimina de la antememoria de sentencias dinámicas y es necesario prepararla de nuevo.

auto Especifica que las vías de acceso se determinan automáticamente por parte del servidor de datos. auto sólo es válido para conexiones con servidores de datos DB2 para z/OS.

-blocking
Especifica el tipo de agrupación en bloque de filas para cursores.

ALL Para los cursores que están especificados por la cláusula FOR READ ONLY o no se especifican como FOR UPDATE, se produce la agrupación en bloque.

UNAMBIG
Para los cursores especificados con la cláusula FOR READ ONLY, se produce la agrupación en bloque.

Para los cursores que no están declarados con la cláusula FOR READ ONLY o FOR UPDATE, no son *ambiguos* y son de *sólo lectura*, se produce la agrupación en bloque. Los cursores *ambiguos* no se agruparán en bloques.

NO No se produce la agrupación en bloque para ningún cursor.

Para conocer la definición de un cursor de sólo lectura y un cursor ambiguo, consulte "DECLARE CURSOR".

-optprofile
Especifica un perfil de optimización que es utilizado para la optimización de sentencias de cambio de datos en los paquetes. Este perfil es un archivo XML que debe existir en el servidor de destino. Si -optprofile no está especificado y el registro especial CURRENT OPTIMIZATION PROFILE está definido, se utiliza el valor de CURRENT OPTIMIZATION PROFILE. Si -optprofile no está especificado, y CURRENT OPTIMIZATION PROFILE no está definido, no se utiliza ningún perfil de optimización.

-optprofile solamente es válido para conexiones con servidores de datos DB2 Database para Linux, UNIX y Windows.

-owner
Especifica el ID de autorización del propietario de los paquetes. El valor por omisión lo establece el servidor de datos.

-owner sólo se aplica a paquetes de IBM Data Server Driver para JDBC y SQLJ.

-sqlid
Especifica un valor en el que se establece el registro especial CURRENT SQLID antes de que DB2Binder ejecute operaciones GRANT en paquetes de IBM Data Server Driver para JDBC y SQLJ. Si el ID de autorización principal no tiene un nivel suficiente de autorización para otorgar privilegios sobre los paquetes y el ID de autorización principal tiene un ID de autorización secundario asociado que tiene estos privilegios, establezca -sqlid en el ID de autorización secundario.

-sqlid solamente es válido para conexiones con servidores de datos DB2 para z/OS.

-generic
Especifica que DB2Binder vuelve a vincular un paquete de usuario en lugar de los paquetes de IBM Data Server Driver para JDBC y SQLJ. Si -generic está especificado, también se deben especificar -action rebind y -package.

-package
Especifica el nombre del paquete que se debe volver a vincular. Esta opción es aplicable a paquetes de usuario. Si -package está especificado, también se deben especificar -action rebind y -generic.

-version
Especifica el ID de versión del paquete que se debe volver a vincular. Si -version está especificado, también se deben especificar -action rebind, -package y -generic.

-keepdynamic
Especifica si el servidor de datos mantiene sentencias de SQL dinámico ya preparadas en la antememoria de sentencias dinámicas después de puntos de confirmación para que esas sentencias preparadas se puedan reutilizar. -keepdynamic se aplica únicamente a las conexiones con DB2 para z/OS. Los valores posibles son:

no El servidor de datos no mantiene sentencias de SQL dinámico ya preparadas en la antememoria de sentencias dinámicas después de puntos de confirmación.

yes El servidor de datos mantiene sentencias de SQL dinámico ya preparadas en la antememoria de sentencias dinámicas después de puntos de confirmación.

No hay valor por omisión para -keepdynamic. Si no se envía un valor al servidor de datos, la configuración del servidor de datos determina si el almacenamiento en antememoria de sentencias dinámicas está activo. La antememoria de sentencias dinámicas solamente está habilitada si el almacenamiento en antememoria de sentencias dinámicas de EDM está habilitado en el servidor de datos. El parámetro de subsistema CACHEDYN debe establecerse en YES para habilitar la antememoria de sentencias dinámicas.

-release
Especifica cuándo liberar los recursos de servidor que un programa utiliza. -release sólo se aplica a las conexiones con DB2 para z/OS. Los valores posibles son:

deallocate
Especifica que los recursos se liberan cuando finaliza un programa. -release deallocate es el valor por omisión de DB2 para z/OS Versión 10 y posterior.

commit
Especifica que los recursos se liberan en los puntos de confirmación. -release commit es el valor por omisión para DB2 para z/OS Versión 9 y anteriores.

-bindoptions
Especifica una serie de caracteres especificada entre comillas. La serie de caracteres está formada uno más pares parámetro-valor que representan opciones para revincular un paquete de usuario. Todos los elementos de la serie están delimitados por espacios:

```
"parm1 valor1 parm2 valor2 ... parmn valorn"
```

-bindoptions no se aplica a paquetes IBM Data Server Driver para JDBC y SQLJ.

Los parámetros y valores posibles son:

bindObjectExistenceRequired
Especifica si el servidor de datos emite un error y no revincula el paquete si no existen todos los objetos o privilegios necesarios en el momento de la revinculación. Los valores posibles son:

true Esta opción corresponde a la opción de vinculación SQLERROR(NOPACKAGE).

false Esta opción corresponde a la opción de vinculación SQLERROR(CONTINUE).

degreeIOParallelism
Especifica si se debe intentar ejecutar consultas estáticas utilizando el proceso en paralelo para maximizar el rendimiento. Los valores posibles son:

1 No se realiza proceso en paralelo.

Esta opción corresponde a la opción de vinculación DEGREE(1).

-1 Permite el proceso en paralelo.

Esta opción corresponde a la opción de vinculación DEGREE(ANY).

packageAuthorizationRules
Determina los valores que se pueden utilizar durante la ejecución para los atributos de SQL dinámico siguientes:

- El ID de autorización que se utiliza para comprobar la autorización
- El calificador que se utiliza para objetos no calificados
- La fuente de opciones de programación de aplicaciones que el servidor de datos utiliza para analizar y verificar semánticamente sentencias de SQL dinámico

- Indicación de si las sentencias de SQL dinámico pueden incluir las sentencias GRANT, REVOKE, ALTER, CREATE, DROP y RENAME

Los valores posibles son:

0 Utilizar comportamiento de ejecución. Éste es el valor por omisión.

Esta opción corresponde a la opción de vinculación DYNAMICRULES(RUN).

1 Utilizar comportamiento de vinculación.

Esta opción corresponde a la opción de vinculación DYNAMICRULES(BIND).

2 Cuando el paquete se ejecuta como o bajo un procedimiento almacenado o función definida por el usuario, el servidor de datos procesa las sentencias de SQL dinámico utilizando el comportamiento de invocación. En otro caso, el servidor de datos procesa las sentencias de SQL dinámico utilizando el comportamiento de ejecución.

Esta opción corresponde a la opción de vinculación DYNAMICRULES(INVOKERUN).

3 Cuando el paquete se ejecuta como o bajo un procedimiento almacenado o función definida por el usuario, el servidor de datos procesa las sentencias de SQL dinámico utilizando el comportamiento de invocación. En otro caso, el servidor de datos procesa las sentencias de SQL dinámico utilizando el comportamiento de vinculación.

Esta opción corresponde a la opción de vinculación DYNAMICRULES(INVOKEBIND).

4 Cuando el paquete se ejecuta como o bajo un procedimiento almacenado o función definida por el usuario, el servidor de datos procesa las sentencias de SQL dinámico utilizando el comportamiento de definición. En otro caso, el servidor de datos procesa las sentencias de SQL dinámico utilizando el comportamiento de ejecución.

Esta opción corresponde a la opción de vinculación DYNAMICRULES(DEFINERUN).

5 Cuando el paquete se ejecuta como o bajo un procedimiento almacenado o función definida por el usuario, el servidor de datos procesa las sentencias de SQL dinámico utilizando el comportamiento de definición. En otro caso, el servidor de datos procesa las sentencias de SQL dinámico utilizando el comportamiento de vinculación.

Esta opción corresponde a la opción de vinculación DYNAMICRULES(DEFINEBIND).

packageOwnerIdentifier
Especifica el ID de autorización del propietario de los paquetes.

isolationLevel
Especifica en qué medida se debe aislar una aplicación respecto a los efectos de otras aplicaciones en ejecución. Los valores posibles son:

1 Lectura no confirmada

Esta opción corresponde a la opción de vinculación ISOLATION(UR).

2 Estabilidad del cursor

Esta opción corresponde a la opción de vinculación ISOLATION(CS).

3 Estabilidad de lectura

Esta opción corresponde a la opción de vinculación ISOLATION(RS).

4 Lectura repetible

Esta opción corresponde a la opción de vinculación ISOLATION(RR).

releasePackageResourcesAtCommit
Especifica cuándo liberar los recursos que un programa utiliza en cada punto de confirmación. Los valores posibles son:

true Esta opción corresponde a la opción de vinculación RELEASE(COMMIT).

false Esta opción corresponde a la opción de vinculación RELEASE(DEALLOCATE).

Si se especifica -bindoptions, se debe especificar también -generic.

-verbose
Especifica que el programa de utilidad DB2Binder muestra información detallada sobre el proceso de enlace.

-help
Especifica que el programa de utilidad DB2Binder describe todas las opciones a las que da soporte. Si se especifica cualquier otra opción con -help, no se tiene en cuenta.

Códigos de retorno de DB2Binder cuando el sistema operativo de destino no es Windows

Si la fuente de datos de destino para DB2Binder no reside en el sistema operativo Windows, DB2Binder devuelve uno de los códigos de retorno siguientes.

Tabla 4. Códigos de retorno de DB2Binder cuando el sistema operativo de destino no es Windows

Código de retorno	Significado
0	Ejecución satisfactoria.
1	Se ha producido un error durante la ejecución de DB2Binder.

Códigos de retorno de DB2Binder cuando el sistema operativo de destino es Windows

Si la fuente de datos de destino para DB2Binder reside en el sistema operativo Windows, DB2Binder devuelve uno de los códigos de retorno siguientes.

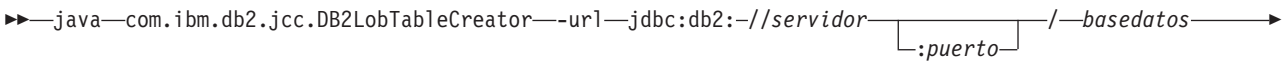
Tabla 5. Códigos de retorno de DB2Binder cuando el sistema operativo de destino es Windows

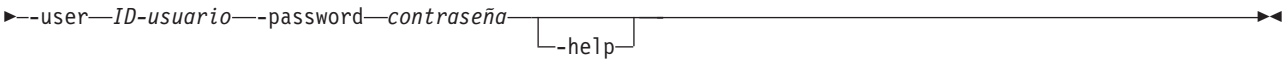
Código de retorno	Significado
0	Ejecución satisfactoria.
-100	No se ha especificado ninguna opción de vinculación.
-101	No se ha especificado el valor de -url.
-102	No se ha especificado el valor de -user.
-103	No se ha especificado el valor de -password.
-200	No se han especificado opciones de vinculación válidas.
-114	La opción -package no se ha especificado, pero la opción -generic sí se ha especificado.
-201	El valor de -url no es válido.
-204	El valor de -action no es válido.
-205	El valor de -blocking no es válido.
-206	El valor de -collection no es válido.
-207	El valor de -dbprotocol no es válido.
-208	El valor de -keepdynamic no es válido.
-210	El valor de -reopt no es válido.
-211	El valor de -size no es válido.
-212	El valor de -tracelevel no es válido.
-307	El valor de -dbprotocol no es compatible con el servidor de datos de destino.
-308	El valor de -keepdynamic no es compatible con el servidor de datos de destino.
-310	El valor de -reopt no es compatible con el servidor de datos de destino.
-313	El valor de -optprofile no es compatible con el servidor de datos de destino.
-401	No se ha encontrado la clase `Binder`.
-402	La conexión con el servidor de datos ha fallado.
-403	La recuperación de DatabaseMetaData para el servidor de datos ha fallado.
-501	No hay más paquetes disponibles en el clúster.
-502	Un paquete existente no es válido.
-503	El proceso de vinculación ha devuelto un error.
-999	Se ha producido un error durante el proceso de una opción de vinculación no documentada.

Programa de utilidad DB2LobTableCreator

El programa de utilidad DB2LobTableCreator crea tablas en un servidor de bases de datos DB2 para z/OS. Esas tablas son necesarias para las aplicaciones JDBC o SQLJ que hacen uso de localizadores de LOB para acceder a datos de columnas DBCLOB o CLOB.

Sintaxis de DB2LobTableCreator





Descripciones de las opciones de DB2LobTableCreator

-url
Especifica la fuente de datos en donde se debe ejecutar DB2LobTableCreator. Las partes variables del valor `-url` son:

jdbc:db2:
Indica que la conexión es con un servidor perteneciente a la familia de productos DB2.

servidor
El nombre de dominio o dirección IP del servidor de bases de datos.

puerto
El número de puerto del servidor TCP/IP que está asignado al servidor de bases de datos. Es un valor entero comprendido entre 0 y 65535. El valor por omisión es 446.

basedatos
Nombre del servidor de bases de datos.

basedatos es el nombre de la ubicación DB2 que se define durante la instalación. Todos los caracteres de este valor deben ser caracteres en mayúsculas. Puede determinar el nombre de ubicación ejecutando la sentencia de SQL siguiente en el servidor:

```
SELECT CURRENT SERVER FROM SYSIBM.SYSDUMMY1;
```

-user
Especifica el ID de usuario utilizado para ejecutar DB2LobTableCreator. Este usuario debe tener autorización para crear tablas en la base de datos DSNATPDB.

-password
Especifica la contraseña del ID de usuario.

-help
Especifica que el programa de utilidad DB2LobTableCreator describe todas las opciones a las que da soporte. Si se especifica cualquier otra opción con -help, no se tiene en cuenta.

Personalización de propiedades de configuración de IBM Data Server Driver para JDBC y SQLJ

Las propiedades de configuración de IBM Data Server Driver para JDBC y SQLJ permiten establecer los valores de las propiedades que tienen un ámbito a nivel de controlador. Estos valores se aplican en las aplicaciones e instancias de DataSource. Puede cambiar los valores sin tener que cambiar el código fuente de aplicación ni las características de DataSource.

Cada valor de propiedad de configuración de IBM Data Server Driver para JDBC y SQLJ tiene este formato:

```
propiedad=valor
```

Puede establecer las propiedades de configuración de estas formas:

- Establezca las propiedades de configuración como propiedades del sistema Java. Los valores de las propiedades de configuración que están definidos como propiedades de sistema de Java omiten los valores de las propiedades de configuración definidos de otras formas.

  En el caso de las aplicaciones Java autónomas, puede establecer las propiedades de configuración como propiedades del sistema Java; para ello, especifique -D*propiedad=valor* para cada propiedad de configuración cuando ejecute el mandato java.
- Establezca las propiedades de configuración en un recurso cuyo nombre se especifica en la propiedad del sistema Java db2.jcc.propertiesFile. Por ejemplo, puede especificar un nombre de vía de acceso absoluta para el valor db2.jcc.propertiesFile.

  En el caso de las aplicaciones Java autónomas, puede establecer las propiedades de configuración especificando la opción -Ddb2.jcc.propertiesFile=*vía_acceso* cuando ejecute el mandato java.
- Establezca las propiedades de configuración en un recurso denominado DB2JccConfiguration.properties. Se utiliza una búsqueda de recursos Java estándar para localizar DB2JccConfiguration.properties. IBM Data Server Driver para JDBC y SQLJ busca este recurso solamente si no ha establecido la propiedad del sistema Java db2.jcc.propertiesFile.

  DB2JccConfiguration.properties puede ser un archivo autónomo o puede estar incluido en un archivo JAR.

  Si el archivo DB2JccConfiguration.properties tiene el esquema de codificación ISO 8859-1 (Latin-1) o si tiene el esquema de codificación Latin-1 con algunos caracteres Unicode (*udddd*), no es necesario realizar la conversión de los caracteres para que IBM Data Server Driver para JDBC y SQLJ pueda utilizar el archivo. Si el archivo DB2JccConfiguration.properties tiene algún otro esquema de codificación, debe utilizar el conversor Java native2ascii para convertir el contenido a Latin-1 o Unicode.

  Si DB2JccConfiguration.properties es un archivo autónomo, la vía de acceso de DB2JccConfiguration.properties debe estar en la concatenación CLASSPATH.

  Si DB2JccConfiguration.properties está en un archivo JAR, el archivo JAR debe estar en la concatenación CLASSPATH.

Configuración especial para acceder a servidores DB2 para z/OS desde programas Java

Si piensa escribir aplicaciones JDBC o SQLJ que acceden a servidores de bases de datos DB2 para z/OS, el proceso de instalación de IBM Data Server Driver para JDBC y SQLJ requiere algunos pasos más.

Acerca de esta tarea

Siga estos pasos para permitir la conectividad con servidores DB2 para z/OS:

Procedimiento

1. Si piensa conectar con servidores de bases de datos DB2 para z/OS Versión 7 o Versión 8, instale estos PTF en esos servidores.

Tabla 6. PTF para procedimientos almacenados de DB2 para z/OS

DB2 para z/OS	Números de PTF o APAR
Versión 7	UQ72083, UQ93889, UK21848

Tabla 6. PTF para procedimientos almacenados de DB2 para z/OS (continuación)

DB2 para z/OS	Números de PTF o APAR
Versión 8	UQ93890, UK21849
Versión 9	PK44166

2. Ejecute el programa de utilidad com.ibm.db2.jcc.DB2Binder para vincular los paquetes DB2 que son utilizados en el servidor por IBM Data Server Driver para JDBC y SQLJ.
3. En los servidores de bases de datos DB2 para z/OS, personalice y ejecute el trabajo DSNTIJMS.

   DSNTIJMS está situado en el archivo *prefijo*.SDSNSAMP. Este trabajo ejecuta las funciones siguientes:

   - Crea los procedimientos almacenados siguientes para habilitar la utilización de métodos DatabaseMetaData, la función de rastreo y el formateo de mensajes de error.
     - SQLCOLPRIVILEGES
     - SQLCOLUMNS
     - SQLFOREIGNKEYS
     - SQLFUNCTIONS
     - SQLFUNCTIONCOLUMNS
     - SQLGETTYPEINFO
     - SQLPRIMARYKEYS
     - SQLPROCEDURECOLS
     - SQLPROCEDURES
     - SQLPSEUDOCOLUMNS (DB2 para z/OS Versión 10 o posterior)
     - SQLSPECIALCOLUMNS
     - SQLSTATISTICS
     - SQLTABLEPRIVILEGES
     - SQLTABLES
     - SQLUDTS
     - SQLCAMESSAGE
   - Crea las tablas siguientes para permitir el almacenamiento eficiente de datos en columnas CLOB o DBCLOB y la utilización de localizadores de LOB para la recuperación de datos CLOB o DBCLOB:
     - SYSIBM.SYSDUMMYU
     - SYSIBM.SYSDUMMYA
     - SYSIBM.SYSDUMMYE

     Una forma alternativa de crear esas tablas es ejecutar el programa de utilidad com.ibm.db2.jcc.DB2LobTableCreator en el cliente para cada uno de los servidores DB2 para z/OS.
4. Habilite el soporte de Unicode para servidores OS/390 y z/OS.

   Si cualquiera de los programas SQLJ o JDBC van a utilizar IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 para conectarse a un servidor DB2 para z/OS Versión 7, el sistema operativo OS/390 o z/OS debe dar soporte al esquema de codificación UTF-8 de Unicode. Este soporte requiere OS/390 Versión 2 Release 9 con el APAR OW44581, o un release posterior de OS/390 o z/OS, además del soporte de OS/390 R8/R9/R10 a Unicode. Los APAR II13048 e II13049 de información contienen datos adicionales.
5. Si tiene previsto utilizar IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 para implementar transacciones distribuidas en

servidores DB2 para z/OS Versión 7, ejecute el programa de utilidad DB2T4XAIndoubtUtil una vez por cada servidor DB2 para z/OS Versión 7.

DB2T4XAIndoubtUtil para transacciones distribuidas con DB2 UDB para los servidores OS/390 y z/OS versión 7

Si piensa implementar transacciones distribuidas utilizando IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 que incluyen servidores DB2 UDB para OS/390 y z/OS Versión 7, es necesario ejecutar el programa de utilidad DB2T4XAIndoubtUtil para esos servidores.

DB2T4XAIndoubtUtil permite que los servidores de la Versión 7, que carecen de soporte interno para transacciones distribuidas que implementan la especificación XA, emulen ese soporte.

DB2T4XAIndoubtUtil ejecuta una de estas tareas o las dos:

- Crea la tabla SYSIBM.INDOUBT y un índice asociado
- Vincula los paquetes T4XAIN01, T4XAIN02, T4XAIN03 y T4XAIN04 de DB2

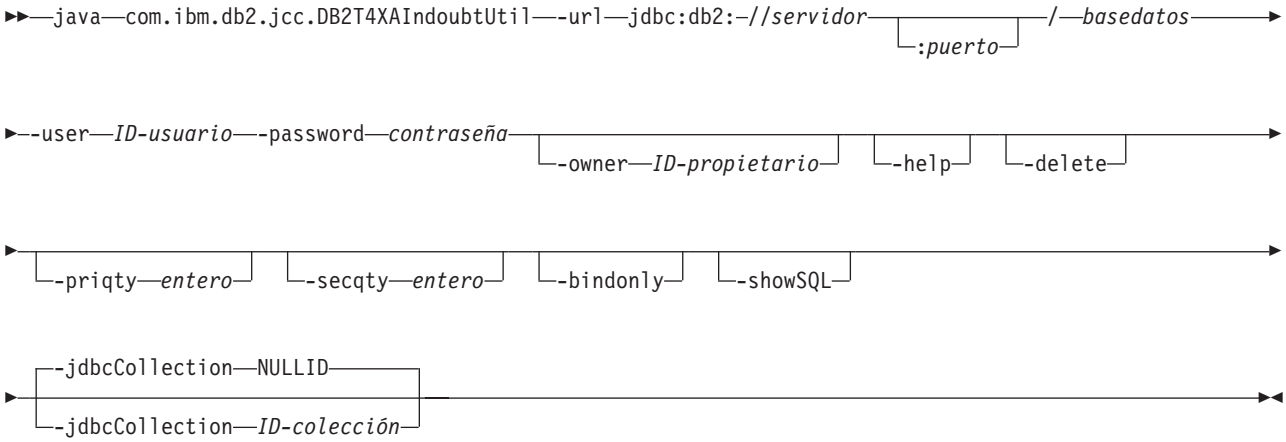
Debe crear y descartar los paquetes T4XAIN01, T4XAIN02, T4XAIN03 y T4XAIN04 sólo mediante la ejecución de DB2T4XAIndoubtUtil. Puede crear y descartar SYSTEM.INDOUBT y su índice manualmente, pero se recomienda que emplee el programa de utilidad. Consulte las Notas de utilización de DB2T4XAIndoubtUtil para obtener instrucciones sobre cómo crear esos objetos manualmente.

Autorización DB2T4XAIndoubtUtil

Si desea ejecutar el programa de utilidad DB2T4XAIndoubtUtil para crear SYSTEM.INDOUBT y vincular los paquetes T4XAIN01, T4XAIN02, T4XAIN03 y T4XAIN04, necesita la autorización SYSADM.

Si desea ejecutar DB2T4XAIndoubtUtil sólo para vincular los paquetes T4XAIN01, T4XAIN02, T4XAIN03 y T4XAIN04, necesita la autorización BIND para los paquetes.

Sintaxis DB2T4XAIndoubtUtil



Información de parámetros de DB2T4XAIndoubtUtil

-url
Especifica la fuente de datos en donde se debe ejecutar DB2T4XAIndoubtUtil. Las partes variables del valor `-url` son:

jdbc:db2:
Indica que la conexión es con un servidor perteneciente a la familia de productos DB2.

servidor
El nombre de dominio o dirección IP del servidor de bases de datos.

puerto
El número de puerto del servidor TCP/IP que está asignado al servidor de bases de datos. Es un valor entero comprendido entre 0 y 65535. El valor por omisión es 446.

basedatos
Nombre del servidor de bases de datos.

basedatos es el nombre de la ubicación DB2 que se define durante la instalación. Todos los caracteres de este valor deben ser caracteres en mayúsculas. Puede determinar el nombre de ubicación ejecutando la sentencia de SQL siguiente en el servidor:

```
SELECT CURRENT SERVER FROM SYSIBM.SYSDUMMY1;
```

-user
Especifica el ID de usuario utilizado para ejecutar DB2T4XAIndoubtUtil. Este usuario debe tener la autorización SYSADM o debe ser miembro de un grupo RACF que se corresponda con un ID de autorización secundaria con la autorización SYSADM.

-password
Especifica la contraseña del ID de usuario.

-owner
Especifica un ID de autorización secundaria que tiene la autorización SYSADM. Utilice el parámetro -owner si -user no tiene la autorización SYSADM. El valor del parámetro -user debe ser miembro de un grupo RACF cuyo nombre es *ID_propietario*.

Cuando se especifique el parámetro -owner, DB2T4XAIndoubtUtil utiliza *ID_propietario* como:

- El ID de autorización para la creación de la tabla SYSIBM.INDOUBT.
- El ID de autorización del propietario de los paquetes T4XAIN01, T4XAIN02, T4XAIN03 y T4XAIN04. Las sentencias de SQL de esos paquetes se ejecutan mediante la autorización de *ID_propietario*.

-help
Especifica que el programa de utilidad DB2T4XAIndoubtUtil describa cada una de las opciones a las que da soporte. Si se especifica cualquier otra opción con -help, no se tiene en cuenta.

-delete
Especifica que el programa de utilidad DB2T4XAIndoubtUtil debe suprimir los objetos que se crearon anteriormente al ejecutar DB2T4XAIndoubtUtil.

-priqty
Especifica la asignación de espacio primaria, en kilobytes, para el espacio de tablas donde reside la tabla SYSIBM.INDOUBT. El valor por omisión de -priqty es 1000.

Importante: El valor de -priqty dividido por el tamaño de página del espacio de tablas donde reside SYSIBM.INDOUBT debe ser mayor que el número máximo de transacciones dudosas que pueden existir en un momento cualquiera. Por ejemplo, para un tamaño de página de 4 KB, el valor por omisión 1000 de -priqty permite alrededor de 250 transacciones dudosas simultáneas.

-secqty
Especifica la asignación de espacio secundaria, en kilobytes, para el espacio de tablas donde reside la tabla SYSIBM.INDOUBT. El valor por omisión de -secqty es 0.

Recomendación: Utilice siempre el valor por omisión 0 para -secqty, y especifique un valor para -priqty que sea lo suficientemente grande para dar cabida al número máximo de transacciones dudosas simultáneas.

-bindonly
Especifica que el programa de utilidad DB2T4XAIndoubtUtil vincule los paquetes T4XAIN01, T4XAIN02, T4XAIN03 y T4XAIN04 y otorgue permiso a PUBLIC para ejecutar los paquetes, pero no crea la tabla SYSIBM.INDOUBT.

-showSQL
Especifica que el programa de utilidad DB2T4XAIndoubtUtil muestre las sentencias de SQL que ejecute.

-jdbcCollection *nombre-colección*|**NULLID**
Especifica el valor del parámetro -collection que se ha utilizado al vincular los paquetes de IBM Data Server Driver para JDBC y SQLJ con el programa de utilidad DB2Binder. El parámetro -jdbcCollection *debe* especificarse si el valor del parámetro -collection especificado de manera implícita o explícita *no* era NULLID.

El valor por omisión es -jdbcCollection NULLID.

Notas de uso de DB2T4XAIndoubtUtil

Para crear manualmente la tabla SYSTEM.INDOUBT y su índice, utilice estas sentencias de SQL:

```
CREATE TABLESPACE INDBTTS
 USING STOGROUP
 LOCKSIZE ROW
 BUFFERPOOL BP0
 SEGSIZE 32
 CCSID EBCDIC;

CREATE TABLE SYSIBM.INDOUBT(indbtXid VARCHAR(140) FOR BIT DATA NOT NULL,
                            uowId VARCHAR(25) FOR BIT DATA NOT NULL,
                            pSyncLog VARCHAR(150) FOR BIT DATA,
                            cSyncLog VARCHAR(150) FOR BIT DATA)
 IN INDBTTS;

CREATE UNIQUE INDEX INDBTIDX ON SYSIBM.INDOUBT(indbtXid, uowId);
```

Ejemplo de DB2T4XAIndoubtUtil

Ejecute la sentencia B2T4XAIndoubtUtil siguiente para permitir que un subsistema DB2 para OS/390 y z/OS Versión 7 cuya dirección IP es mvs1, su número de puerto es 446, y su nombre de ubicación DB2 es SJCEC1, participe en transacciones distribuidas XA.

```
java com.ibm.db2.jcc.DB2T4XAIndoubtUtil -url jdbc:db2://mvs1:446/SJCEC1 \
  -user SYSADM -password mypass
```

Capítulo 3. Programación de aplicaciones JDBC

La escritura de una aplicación JDBC tiene mucho en común con la escritura de una aplicación SQL en cualquier otro lenguaje.

En general, es necesario que realice las acciones siguientes:
- Acceda a los paquetes de Java donde residen los métodos JDBC.
- Declare variables para enviar datos a tablas de DB2 o recuperar datos de ellas.
- Conecte con una fuente de datos.
- Ejecute sentencias de SQL.
- Trate los errores y avisos de SQL.
- Desconecte de la fuente de datos.

Aunque las tareas que necesita realizar son similares a las que se ejecutan en otros lenguajes, la forma de ejecutarlas es algo diferente.

Ejemplo de una aplicación JDBC simple

Aplicación JDBC simple que muestra los elementos básicos que es necesario incluir en una aplicación JDBC.

Figura 1. Aplicación JDBC sencilla

```
import java.sql.*;                                                    1

public class EzJava
{
  public static void main(String[] args)
  {
    String urlPrefix = "jdbc:db2:";
    String url;
    String user;
    String password;
    String empNo;                                                     2
    Connection con;
    Statement stmt;
    ResultSet rs;

    System.out.println ("**** Especificar clase EzJava");

    // Comprobar que el primer argumento tenga el formato correcto para
    // la parte del URL jdbc: db2:,
    // tal como se describe en el tema Conexión a una fuente
    // de datos utilizando la interfaz DriverManager
    // con IBM Data Server Driver para JDBC y SQLJ.
    // Por ejemplo, para IBM Data Server Driver para
    // conectividad de JDBC y SQLJ de tipo 2,
    // args[0] puede ser MVS1DB2M. Para la
    // conectividad de tipo 4, args[0] podría ser
    // stlmvs1:10110/MVS1DB2M.

    if (args.length!=3)
    {
      System.err.println ("Valor no válido. Primer argumento añadido a "+
       "jdbc:db2: debe especificar un URL válido.");
      System.err.println ("El segundo argumento debe ser un ID de usuario válido.");
      System.err.println ("El tercer argumento debe ser la contraseña del ID de usuario.");
      System.exit(1);
    }
    url = urlPrefix + args[0];
    user = args[1];
    password = args[2];
    try
```

```
    {
      // Cargar el controlador
      Class.forName("com.ibm.db2.jcc.DB2Driver");                          3a
      System.out.println("**** Controlador JDBC cargado");

      // Crear conexión utilizando IBM Data Server Driver para JDBC y SQLJ
      con = DriverManager.getConnection (url, user, password);             3b
      // Confirmar los cambios manualmente
      con.setAutoCommit(false);
      System.out.println("**** Creada una conexión JDBC con la fuente de datos");

      // Crear el objeto Statement
      stmt = con.createStatement();                                        4a
      System.out.println("**** Creado el objeto Statement de JDBC");

      // Ejecutar una consulta y generar instancia del conjunto de resultados
      rs = stmt.executeQuery("SELECT EMPNO FROM EMPLOYEE");                4b
      System.out.println("**** Creado el objeto JDBC ResultSet");

      // Imprimir todos los números de empleado en el dispositivo de salida estándar
      while (rs.next()) {
        empNo = rs.getString(1);
        System.out.println("Número de empleado = " + empNo);
      }
      System.out.println("**** Buscadas todas las filas del conjunto resultados JDBC");
      // Cerrar el conjunto de resultados
      rs.close();
      System.out.println("**** Cerrado el conjunto de resultados de JDBC");

      // Cerrar el objeto Statement
      stmt.close();
      System.out.println("**** Cerrado el objeto Statement de JDBC");

      // La conexión debe estar en un límite de unidad trabajo para permitir cierre
      con.commit();
      System.out.println ( "**** Transacción confirmada" );

      // Cierre la conexión
      con.close();                                                         6
      System.out.println("**** Desconectado de la fuente de datos");

      System.out.println("**** Salida de JDBC de la clase EzJava - sin errores");

    }

    catch (ClassNotFoundException e)
    {
      System.err.println("No se pudo cargar el controlador JDBC");
      System.out.println("Exception: " + e);
      e.printStackTrace();
    }

    catch(SQLException ex)                                                 5
    {
      System.err.println("Información sobre SQLException");
      while(ex!=null) {
        System.err.println ("Mensaje de error: " + ex.getMessage());
        System.err.println ("SQLSTATE: " + ex.getSQLState());
        System.err.println ("Código de error: " + ex.getErrorCode());
        ex.printStackTrace();
        ex = ex.getNextException(); // Para controladores que soportan
                                    // excepciones encadenadas
      }
    }
  }  // Fin main
}    // Fin EzJava
```

Notas para la Figura 1 en la página 27:

Nota	Descripción
1	Esta sentencia importa el paquete java.sql, el cual contiene la API básica de JDBC. Para obtener información sobre otros paquetes Java que puede ser necesario acceder, consulte "Paquetes Java para soporte de JDBC".

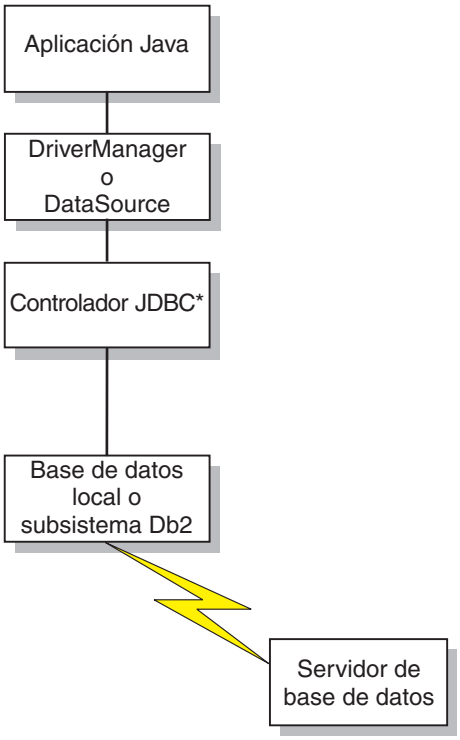
Nota	Descripción
2	La variable empNo de tipo `String` realiza la función de una variable del lenguaje principal. Es decir, se utiliza para contener datos obtenidos en una consulta de SQL. Consulte "Variables en aplicaciones JDBC" para obtener más información.
3a y 3b	Estos dos conjuntos de sentencias muestran cómo conectar con una fuente de datos utilizando una de dos interfaces disponibles. Consulte "Cómo las aplicaciones JDBC conectan con una fuente de datos" para conocer más detalles. El paso 3a (carga del controlador JDBC) no es necesario si se utiliza JDBC 4.0 o posterior.
4a y 4b	Estos dos conjuntos de sentencias muestran cómo ejecutar una operación SELECT en JDBC. Para obtener información sobre cómo realizar otras operaciones de SQL, consulte "Interfaces de JDBC para ejecutar SQL".
5	Este bloque `try/catch` muestra el uso de la clase SQLException para el manejo de errores de SQL. Para obtener más información sobre el manejo de errores de SQL, consulte "Manejo de una excepción de SQL cuando se utiliza IBM Data Server Driver para JDBC y SQLJ". Para obtener información sobre el manejo de avisos de SQL, consulte "Manejo de un aviso de SQL cuando se utiliza IBM Data Server Driver para JDBC y SQLJ".
6	Esta sentencia desconecta la aplicación respecto de la fuente de datos. Consulte "Desconexión de fuente de datos en aplicaciones JDBC".

Conexión de las aplicaciones JDBC a una fuente de datos

Para poder ejecutar sentencias de SQL en un programa SQL cualquiera, debe estar conectado con una fuente de datos.

IBM Data Server Driver para JDBC y SQLJ es compatible con la conectividad de tipo 2 y tipo 4. Las conexiones con bases de datos DB2 pueden utilizar conectividad de tipo 2 o tipo 4. Las conexiones con bases de datos IBM Informix pueden utilizar conectividad de tipo 4.

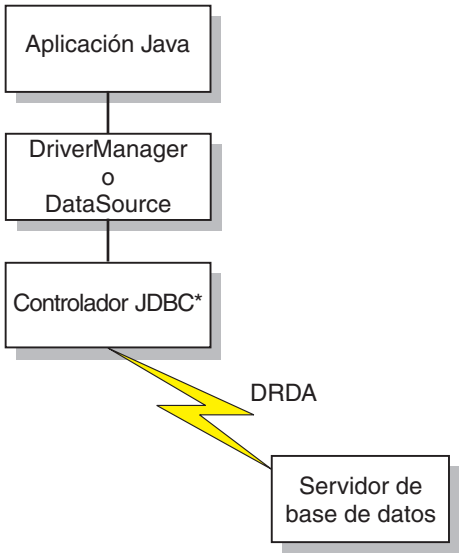
La figura siguiente muestra cómo una aplicación Java se conecta a una fuente de datos mediante IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2.



*Código de bytes de Java ejecutados bajo JVM
y código nativo

Figura 2. Flujo de una aplicación Java para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2

La figura siguiente muestra cómo una aplicación Java se conecta a una fuente de datos mediante IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4.



*Código de bytes de Java ejecutado bajo JVM

Figura 3. Flujo de una aplicación Java para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4

Conexión con una fuente de datos utilizando la interfaz DriverManager con IBM Data Server Driver para JDBC y SQLJ

Una aplicación JDBC puede establecer una conexión con una fuente de datos utilizando la interfaz DriverManager de JDBC, la cual forma parte del paquete java.sql.

Acerca de esta tarea

Estos son los pasos para establecer una conexión:

Procedimiento

1. Cargue el controlador JDBC invocando el método Class.forName.

   Si va a utilizar JDBC 4.0 o posterior, no tiene que cargar de forma explícita el controlador JDBC.

   Para IBM Data Server Driver para JDBC y SQLJ, el controlador se carga invocando el método Class.forName con este argumento:

   ```
   com.ibm.db2.jcc.DB2Driver
   ```

   Para mantener la compatibilidad con controladores JDBC anteriores, puede utilizar el argumento siguiente como alternativa:

   ```
   COM.ibm.db2os390.sqlj.jdbc.DB2SQLJDriver
   ```

   El código de programa siguiente muestra la carga de IBM Data Server Driver para JDBC y SQLJ:

   ```
   try {
     // Cargar IBM Data Server Driver para JDBC y SQLJ mediante DriverManager
     Class.forName("com.ibm.db2.jcc.DB2Driver");
   } catch (ClassNotFoundException e) {
       e.printStackTrace();
   }
   ```

El bloque `catch` se utiliza para imprimir un error si se no se encuentra el controlador.

2. Conecte con una fuente de datos invocando el método DriverManager.getConnection.

   Puede utilizar uno de los formatos siguientes de getConnection:

   ```
   getConnection(String url);
   getConnection(String url, usuario, contraseña);
   getConnection(String url, java.util.Properties info);
   ```

   Para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4, el método getConnection debe especificar un ID de usuario y una contraseña mediante parámetros o valores de propiedad.

   El argumento *url* representa una fuente de datos e indica el tipo de conectividad JDBC que se está utilizando.

   El argumento *info* es un objeto de tipo java.util.Properties que contiene un conjunto de propiedades de controlador correspondientes a la conexión. Especificar el argumento *info* es una alternativa a especificar series de caracteres *propiedad=valor;* en el URL. Consulte "Propiedades de IBM Data Server Driver para JDBC y SQLJ" para conocer las propiedades que puede especificar.

   Existen varias maneras de especificar un ID de usuario y una contraseña para una conexión:

   - Utilice la modalidad del método getConnection donde se especifica el *url* con cláusulas `propiedad=valor;`, e incluya las propiedades correspondientes al usuario y la contraseña en el URL.
   - Utilice el formato del método getConnection en el que se especifica el *usuario* y *contraseña*.
   - Utilice la modalidad del método getConnection donde se especifica *info*, después de establecer las propiedades correspondientes al usuario y la contraseña en un objeto java.util.Properties.

Ejemplo

Ejemplo: establecimiento de una conexión y especificación del ID de usuario y la contraseña en un URL:

```
String url = "jdbc:db2://myhost:5021/mydb:" +
  "user=dbadm;password=dbadm;";

                                        // Definir URL para fuente de datos
Connection con = DriverManager.getConnection(url);
                                        // Crear conexión
```

Ejemplo: establecimiento de una conexión y especificación del ID de usuario y la contraseña en parámetros de usuario y contraseña:

```
String url = "jdbc:db2://myhost:5021/mydb";
                                        // Definir URL para fuente de datos
String user = "dbadm";
String password="dbadm";
Connection con = DriverManager.getConnection(url, user, password);
                                        // Crear conexión
```

Ejemplo: establecimiento de una conexión y especificación del ID de usuario y la contraseña en un objeto java.util.Properties:

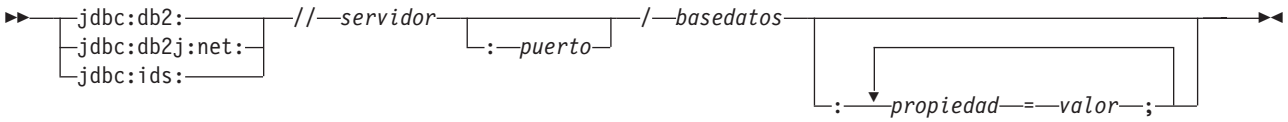
```
Properties properties = new Properties(); // Crear objeto Properties
properties.put("user", "dbadm");         // Definir ID de usuario para la conexión
properties.put("password", "dbadm");     // Definir contraseña para la conexión
String url = "jdbc:db2://myhost:5021/mydb";
```

```
                                                // Definir URL para fuente de datos
Connection con = DriverManager.getConnection(url, properties);
                                                // Crear conexión
```

Formato del URL para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4

Si está utilizando la conectividad de tipo 4 en su aplicación JDBC y crea una conexión mediante la interfaz DriverManager, debe especificar un URL en la llamada a DriverManager.getConnection que indique el uso de la conectividad de tipo 4.

IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4Sintaxis del URL



IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4Descripciones de las opciones para el URL

Los componentes del URL tienen los significados siguientes:

jdbc:db2: o jdbc:db2j:net:
Los significados de la parte inicial del URL son los siguientes:

jdbc:db2:
Indica que la conexión es con un servidor DB2 para z/OS, DB2 Database para Linux, UNIX y Windows.

También se puede utilizar jdbc:db2: para conexión con una base de datos IBM Informix, para permitir la portabilidad de aplicaciones.

jdbc:db2j:net:
Indica que la conexión es con un servidor IBM Cloudscape remoto.

jdbc:ids:
Indica que la conexión es con una fuente de datos IBM Informix. jdbc:informix-sqli: también indica que la conexión es con una fuente de datos IBM Informix, sin embargo, se debería utilizar jdbc:ids:

servidor
Nombre de dominio o dirección IP de la fuente de datos.

puerto
Número de puerto del servidor TCP/IP que está asignado a la fuente de datos. Es un valor entero comprendido entre 0 y 65535. El valor por omisión es 446.

basedatos
Nombre de la fuente de datos.

- Si la conexión es con un servidor DB2 para z/OS, *basedatos* es el nombre de ubicación de DB2 que se define durante la instalación. Todos los caracteres del nombre de ubicación de DB2 deben estar en mayúsculas. IBM Data Server Driver para JDBC y SQLJ no convierte en mayúsculas para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 los caracteres de la base de datos que están en minúsculas.

  Puede determinar el nombre de ubicación ejecutando la sentencia de SQL siguiente en el servidor:

```
SELECT CURRENT SERVER FROM SYSIBM.SYSDUMMY1;
```

- Si la conexión es con un servidor DB2 para z/OS o un servidor DB2 para i, todos los caracteres de *basedatos* deben estar en mayúsculas.
- Si la conexión es con un servidor DB2 Database para Linux, UNIX y Windows, *basedatos* es el nombre de la base de datos que se define durante la instalación.
- Si la conexión es con un servidor IBM Informix, *basedatos* es el nombre de la base de datos. El nombre no es sensible a las mayúsculas y las minúsculas. El servidor convierte el nombre a minúsculas.
- Si la conexión es con un servidor IBM Cloudscape, *basedatos* es el nombre totalmente calificado del archivo donde reside la base de datos. Este nombre se debe incluir entre comillas dobles ("). Por ejemplo:

  ```
  "c:/basedatos/testdb"
  ```

propiedad=valor;
: Propiedad y su valor para la conexión JDBC. Puede especificar uno o más pares propiedad-valor. Cada par propiedad-valor, incluido el último, debe terminar con un signo de punto y coma (;). No incluye espacios ni otros caracteres de espacio en blanco dentro de la lista de series de propiedad y valor.

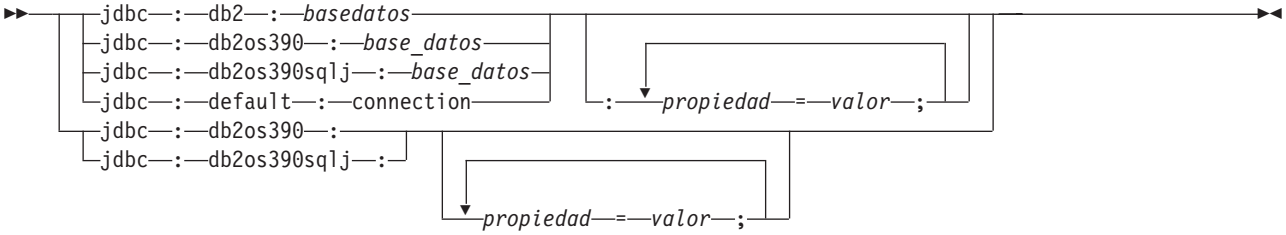
  Algunas propiedades con un tipo de datos int tienen valores de campo constante predefinidos. Hay que resolver los valores de campo constante predefinidos con sus valores enteros antes de utilizar dichos valores en el parámetro *url*. Por ejemplo, no se puede utilizar com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL en un parámetro *url*. Sin embargo, se puede crear una serie URL que incluya com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL, y asignar la serie URL a una variable String. A continuación, se puede utilizar la variable String en el parámetro *url*:

  ```
  String url =
   "jdbc:db2://sysmvs1.stl.ibm.com:5021/STLEC1" +
   ":user=dbadm;password=dbadm;" +
   "traceLevel=" +
   (com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL) + ";";
   Connection con =
     java.sql.DriverManager.getConnection(url);
  ```

Formato del URL para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2

Si está utilizando la conectividad de tipo 2 en su aplicación JDBC y crea una conexión mediante la interfaz DriverManager, debe especificar un URL en la llamada a DriverManager.getConnection que indique el uso de la conectividad de tipo 2.

IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2Sintaxis del URL



IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2Descripciones de las opciones para el URL

Los componentes del URL tienen los significados siguientes:

jdbc:db2: o jdbc:db2os390: o jdbc:db2os390sqlj: o jdbc:default:connection
Los significados de la parte inicial del URL son los siguientes:

jdbc:db2: o jdbc:db2os390: o jdbc:db2os390sqlj:
Indica que la conexión es con un servidor DB2 para z/OS o DB2 Database para Linux, UNIX y Windows. jdbc:db2os390: y jdbc:db2os390sqlj: se utilizan para mantener la compatibilidad de programas que se grabaron para controladores más antiguos y sólo se aplican a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 para DB2 para z/OS.

jdbc:default:connection
Indica que el URL es para una conexión con el subsistema local mediante una hebra DB2 que está controlada por CICS o IMS o por el entorno de procedimiento almacenado de Java

basedatos
Nombre del servidor de bases de datos.

- *basedatos* es el nombre de base de datos que se define durante la instalación, si el valor de la propiedad de conexión serverName es nulo. Si el valor de la propiedad serverName no es nulo, *basedatos* es un alias de base de datos.

propiedad=valor;
Propiedad y su valor para la conexión JDBC. Puede especificar uno o más pares propiedad-valor. Cada par propiedad-valor, incluido el último, debe terminar con un signo de punto y coma (;). No incluye espacios ni otros caracteres de espacio en blanco dentro de la lista de series de propiedad y valor.

Algunas propiedades con un tipo de datos int tienen valores de campo constante predefinidos. Hay que resolver los valores de campo constante predefinidos con sus valores enteros antes de utilizar dichos valores en el parámetro *url*. Por ejemplo, no se puede utilizar com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL en un parámetro *url*. Sin embargo, se puede crear una serie URL que incluya com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL, y asignar la serie URL a una variable String. A continuación, se puede utilizar la variable String en el parámetro *url*:

```
String url =
 "jdbc:db2:STLEC1" +
 ":user=dbadm;password=dbadm;" +
 "traceLevel=" +
 (com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL) + ";";
 Connection con =
   java.sql.DriverManager.getConnection(url);
```

Conexión con una fuente de datos mediante la interfaz DataSource

Si es necesario que sus aplicaciones se puedan migrar de una fuente de datos a otra, debe utilizar la interfaz DataSource.

Acerca de esta tarea

El uso de DriverManager para conectar con una fuente de datos reduce la portabilidad, pues la aplicación debe identificar un nombre de clase y URL determinados para el controlador JDBC. El nombre de clase y el URL del controlador son específicos de un proveedor de JDBC, de la implementación del controlador y de la fuente de datos.

Cuando se conecta a una fuente de datos utilizando la interfaz DataSource, utiliza un objeto DataSource.

La forma más sencilla de utilizar un objeto DataSource es crear y utilizar el objeto en la misma aplicación, tal como hace en la interfaz DriverManager. Sin embargo, este método no proporciona portabilidad.

La mejor forma de utilizar un objeto DataSource es que el administrador del sistema cree y gestione el objeto por separado, utilizando WebSphere Application Server o alguna otra herramienta. El programa por el que se crea y gestiona un objeto DataSource también utiliza Java Naming and Directory Interface (JNDI) para asignar un nombre lógico al objeto DataSource. La aplicación JDBC que hace uso del objeto DataSource puede luego referirse al objeto utilizando su nombre lógico, y no necesita ninguna información sobre la fuente de datos subyacente. Además, el administrador del sistema puede modificar los atributos de la fuente de datos y el usuario no necesita cambiar su programa de aplicación.

Para obtener más información sobre cómo utilizar WebSphere para desplegar objetos DataSource, diríjase al URL siguiente de la web:

```
http://www.ibm.com/software/webservers/appserv/
```

Para conocer cómo desplegar objetos DataSource por sí mismo, consulte "Creación y despliegue de objetos DataSource".

Puede utilizar la interfaz DataSource y la interfaz DriverManager en la misma aplicación, pero para lograr una portabilidad máxima es recomendable que utilice solo la interfaz DataSource para obtener conexiones.

Para obtener una conexión utilizando un objeto DataSource que creó el administrador del sistema y al que éste asignó un nombre lógico, siga estos pasos:

Procedimiento

1. Consulte al administrador del sistema para obtener el nombre lógico de la fuente de datos con la que necesita conectar.
2. Cree un objeto Context para utilizarlo en el paso siguiente. La interfaz Context forma parte de Java Naming and Directory Interface (JNDI) y no de JDBC.
3. En su programa de aplicación, utilice JNDI para obtener el objeto DataSource que está asociado al nombre lógico de la fuente de datos.
4. Utilice el método DataSource.getConnection para obtener la conexión.

   Puede utilizar uno de los formatos siguientes del método getConnection:

   ```
   getConnection();
   getConnection(String usuario, String contraseña);
   ```

   Utilice la segunda forma si necesita especificar un ID de usuario y una contraseña para la conexión que sean diferentes de los que se especificaron al desplegar el objeto DataSource.

Ejemplo

Ejemplo de obtención de una conexión utilizando un objeto DataSource que fue creado por el administrador del sistema: en este ejemplo, el nombre lógico de la fuente de datos con la que desea conectar es `jdbc/sampledb`. Los números situados a la derecha de determinadas sentencias corresponden a pasos descritos anteriormente.

```
import java.sql.*;
import javax.naming.*;
import javax.sql.*;
...
Context ctx=new InitialContext();                          2
DataSource ds=(DataSource)ctx.lookup("jdbc/sampledb");     3
Connection con=ds.getConnection();                         4
```

Figura 4. Obtención de una conexión utilizando un objeto DataSource

Ejemplo de creación y utilización de un objeto DataSource en la misma aplicación:

Figura 5. Creación y utilización de un objeto DataSource en la misma aplicación

```
import java.sql.*;          // Base JDBC
import javax.sql.*;        // Métodos para JDBC
import com.ibm.db2.jcc.*; // Interfaces de IBM Data Server Driver   1
                          // para JDBC y SQLJ
DB2SimpleDataSource dbds=new DB2SimpleDataSource();                2
dbds.setDatabaseName("dbloc1");                                    3
                          // Asignar el nombre de ubicación
dbds.setDescription("Our Sample Database");
                          // Descripción de la documentación
dbds.setUser("john");
                          // Asignar el ID de usuario
dbds.setPassword("dbadm");
                          // Asignar la contraseña
Connection con=dbds.getConnection();                               4
                          // Crear un objeto Connection
```

Nota	Descripción
1	Importa el paquete donde reside la implementación de la interfaz DataSource.
2	Crea un objeto DB2SimpleDataSource. DB2SimpleDataSource es una de las implementaciones para IBM Data Server Driver para JDBC y SQLJ de la interfaz DataSource. Consulte "Creación y despliegue de objetos DataSource" para obtener información sobre las implementaciones de DataSource de DB2.
3	Los métodos setDatabaseName, setDescription, setUser y setPassword asignan atributos al objeto DB2SimpleDataSource. Consulte "Propiedades del IBM Data Server Driver para JDBC y SQLJ" para conocer los atributos que puede definir para un objeto DB2SimpleDataSource cuando se utiliza IBM Data Server Driver para JDBC y SQLJ.
4	Establece una conexión con la fuente de datos representada por el objeto DB2SimpleDataSource.

Cómo determinar qué tipo de conectividad de IBM Data Server Driver para JDBC y SQLJ utilizar

IBM Data Server Driver para JDBC y SQLJ da soporte a dos tipos de conectividad: la conectividad de tipo 2 y de tipo 4.

En el caso de la interfaz DriverManager, el tipo de conectividad se especifica a través del URL del método DriverManager.getConnection. En el caso de la interfaz DataSource, el tipo de conectividad se especifica a través de la propiedad `driverType`.

En la tabla siguiente, se resumen las diferencias entre las conectividades de tipo 2 y tipo 4:

Tabla 7. Comparación de IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 y IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4

Función	Soporte de IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2	Soporte de IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4
Equilibrado de la carga de trabajo de Sysplex y concentrador de conexiones	Soportados mediante DB2 Connect	Soportados directamente por el controlador para una conexión de una sola JVM. Soportados mediante DB2 Connect en todas las JVM.
Protocolos de comunicación	TCP/IP	TCP/IP
Rendimiento	Mejor para acceder a un servidor DB2 local	Mejor para acceder a un servidor DB2 remoto
Instalación	Requiere la instalación de bibliotecas nativas y de clases Java.	Requiere la instalación de clases Java solamente.
Procedimientos almacenados	Se pueden utilizar para llamar o ejecutar procedimientos almacenados.	Se pueden utilizar sólo para llamar a procedimientos almacenados.
Proceso de transacciones distribuidas (XA)	Soportado	Soportado
Compatibilidad J2EE 1.4	Compatible	Compatible

Los puntos siguientes pueden resultar de ayuda a la hora de determinar el tipo de conectividad que debe utilizarse.

Utilice IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en las circunstancias siguientes:

- La aplicación JDBC o SQLJ se ejecuta localmente la mayor parte del tiempo.

  El rendimiento de las aplicaciones locales mejora con la conectividad de tipo 2.
- Está *ejecutando* un procedimiento almacenado de Java.

  El entorno de un procedimiento almacenado consta de dos partes: un programa cliente, desde el cual se realizan llamadas a procedimientos almacenados, y un programa de servidor, donde se encuentran los procedimientos almacenados. Se puede llamar a un procedimiento almacenado de un programa JDBC o SQLJ que utilice conectividad de tipo 2 o tipo 4; sin embargo, para ejecutar un procedimiento almacenado de Java, deberá utilizar la conectividad de tipo 2.

Utilice IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 en las circunstancias siguientes:

- La aplicación JDBC o SQLJ se ejecuta de forma remota la mayor parte del tiempo.

  El rendimiento de las aplicaciones remotas mejora con la conectividad de tipo 4.
- Está utilizando el soporte de concentrador de conexiones y equilibrado de la carga de trabajo de Sysplex de IBM Data Server Driver para JDBC y SQLJ.

Objetos de conexión JDBC

Cuando se conecta a una fuente de datos mediante cualquiera de los dos métodos de conexión, crea un objeto Connection, que representa la conexión con la fuente de datos.

Utilice este objeto Connection para realizar lo siguiente:

- Crear objetos Statement, PreparedStatement, y CallableStatement para ejecutar sentencias de SQL. Este tema se trata en "Ejecución de sentencias de SQL en aplicaciones JDBC".
- Reunir información sobre la fuente de datos a la que está conectado. Este proceso se describe en "Conocer sobre una fuente de datos utilizando métodos DatabaseMetaData".
- Confirmar o retrotraer transacciones. Puede confirmar transacciones de forma manual o automática. Estas operaciones se describen en "Confirmar o retrotraer una transacción JDBC".
- Cerrar la conexión con la fuente de datos. Esta operación se describe en "Desconexión de fuentes de datos en aplicaciones JDBC".

Creación y despliegue de objetos DataSource

A partir de la versión 2.0, JDBC proporciona la interfaz DataSource para conectar con una fuente de datos. La utilización de la interfaz DataSource es la forma preferida de conectar con una fuente de datos.

Acerca de esta tarea

La utilización de la interfaz DataSource comprende dos etapas:

- Crear y desplegar objetos DataSource. Normalmente, esto lo hace un administrador del sistema utilizando una herramienta como WebSphere Application Server.
- Utilizar objetos DataSource para crear una conexión. Esto se realiza en el programa de aplicación.

Este tema contiene información necesaria para que el propio usuario pueda crear y desplegar objetos DataSource.

IBM Data Server Driver para JDBC y SQLJ proporciona las siguientes implementaciones de DataSource:

- com.ibm.db2.jcc.DB2SimpleDataSource, que no es compatible con la agrupación de conexiones. Puede utilizar esta implementación con IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 o IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4.
- com.ibm.db2.jcc.DB2ConnectionPoolDataSource, que es compatible con la agrupación de conexiones. Puede utilizar esta implementación con IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 o IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4.
- com.ibm.db2.jcc.DB2XADataSource, que da soporte a la agrupación de conexiones y a las transacciones distribuidas. La agrupación de conexiones es proporcionada por WebSphere Application Server u otro servidor de aplicaciones. Puede utilizar esta implementación solamente con IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4.

Cuando crea y despliega un objeto DataSource, debe efectuar estas tareas:

1. Crear una instancia de la implementación de DataSource apropiada.

2. Establecer las propiedades del objeto DataSource.
3. Registrar el objeto en el servicio de denominación de Java Naming and Directory Interface (JNDI).

El ejemplo siguiente muestra cómo realizar estas tareas.

```
import java.sql.*;          // Base JDBC
import javax.naming.*;     // Servicios de denominación de JNDI
import javax.sql.*;        // Métodos adicionales para JDBC
import com.ibm.db2.jcc.*; // Implementación de IBM Data
                           // Server para JDBC y SQLJ
                           // de JDBC
                           // estándar de JDBC

DB2SimpleDataSource dbds = new com.ibm.db2.jcc.DB2SimpleDataSource();   1

dbds.setDatabaseName("db2loc1");                                        2
dbds.setDescription("Our Sample Database");
dbds.setUser("john");
dbds.setPassword("mypw");
...
Context ctx=new InitialContext();                                       3
Ctx.bind("jdbc/sampledb",dbds);                                         4
```

Figura 6. Ejemplo de creación y despliegue de un objeto DataSource

Nota	Descripción
1	Crea una instancia de la clase DB2SimpleDataSource.
2	Esta sentencia y las tres sentencias siguientes definen valores para propiedades del objeto DB2SimpleDataSource.
3	Crea un contexto para su utilización por JNDI.
4	Asocia el objeto dbds de DBSimple2DataSource con el nombre lógico jdbc/sampledb. Una aplicación que haga uso de este objeto puede hacer referencia a él utilizando el nombre jdbc/sampledb.

Paquetes Java para el soporte JDBC

Para invocar métodos de JDBC necesita poder acceder a todos los paquetes Java (o parte de ellos) donde residen estos métodos.

Puede hacerlo importando los paquetes o clases específicas, o bien utilizando los nombres de clase totalmente calificados. Puede necesitar los paquetes o clases siguientes para su programa de JDBC:

java.sql
Contiene la API básica de JDBC.

javax.naming
Contiene clases e interfaces para Java Naming and Directory Interface (JNDI), que se suele utilizar para implementar una DataSource (fuente de datos).

javax.sql
Contiene métodos para producir aplicaciones de servidor mediante Java

com.ibm.db2.jcc
Contiene la implementación de JDBC para IBM Data Server Driver para JDBC y SQLJ.

Obtención de información acerca de una fuente de datos mediante métodos DatabaseMetaData

La interfaz DatabaseMetaData contiene métodos para recuperar información sobre una fuente de datos. Estos métodos son útiles cuando escribe aplicaciones genéricas que pueden acceder a diversas fuentes de datos.

Acerca de esta tarea

En las aplicaciones genéricas que pueden acceder a diversas fuentes de datos, debe comprobar si una fuente de datos puede manejar diversas operaciones de base de datos antes de ejecutarlas. Por ejemplo, debe determinar si el controlador de una fuente de datos se encuentra en el nivel JDBC 3.0 antes de invocar métodos JDBC 3.0 para ese controlador.

Los métodos de DatabaseMetaData proporcionan los tipos de información siguientes:

- Características soportadas por la fuente de datos, tales como el nivel SQL de ANSI
- Información específica sobre el controlador JDBC, tal como el nivel del controlador
- Límites, tales como el número máximo de columnas que puede tener un índice
- Indicación de si la fuente de datos soporta sentencias de definición de datos (CREATE, ALTER, DROP, GRANT, REVOKE)
- Lista de objetos contenidos en la fuente de datos, tales como tablas, índices o procedimientos
- Indicación de si la fuente de datos soporta diversas funciones JDBC, tales como las actualizaciones por lotes o los conjuntos de resultados desplazables ResultSet
- Lista de funciones escalares soportadas por el controlador

Para invocar métodos DatabaseMetaData, siga estos pasos básicos:

Procedimiento

1. Cree un objeto DatabaseMetaData invocando el método getMetaData para la conexión.
2. Invoque métodos de DatabaseMetaData para obtener información sobre la fuente de datos.
3. Si el método devuelve un ResultSet:
   a. En un bucle, posicione el cursor utilizando el método next y recupere datos de cada columna de la fila actual del objeto ResultSet utilizando métodos getXXX.
   b. Invoque el método close para cerrar el objeto ResultSet.

Ejemplo

Ejemplo: el código de programa siguiente muestra cómo utilizar métodos DatabaseMetaData para determinar la versión del controlador, obtener una lista de los procedimientos almacenados existentes en la fuente de datos, y obtener una lista de funciones de fecha y hora compatibles con el controlador. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

Figura 7. Uso de métodos DatabaseMetaData para obtener información sobre una fuente de datos

```
Connection con;
DatabaseMetaData dbmtadta;
ResultSet rs;
int mtadtaint;
String procSchema;
String procName;
String dtfnList;
...
dbmtadta = con.getMetaData();     // Crear el objeto DatabaseMetaData  1
mtadtaint = dmtadta.getDriverVersion();                                 2
                                  // Comprobar la versión del controlador
System.out.println("Versión de controlador: " + mtadtaint);
rs = dbmtadta.getProcedures(null, null, "%");
                                  // Obtener información de todos los procedimientos
while (rs.next()) {               // Situar el cursor                   3a
 procSchema = rs.getString("PROCEDURE_SCHEM");
                                  // Obtener el esquema del procedimiento
 procName = rs.getString("PROCEDURE_NAME");
                                  // Obtener el nombre del procedimiento
 System.out.println(procSchema + "." + procName);
                                  // Imprimir nombre de procedimiento calificado
}
dtfnList = dbmtadta.getTimeDateFunctions();
                                  // Obtener lista de las funciones
                                  // de fecha y hora permitidas
System.out.println("Funciones de fecha y hora soportadas:");
System.out.println(dtfnList); // Imprimir la lista de las funciones
                              // de fecha y hora
rs.close();                   // Cerrar el conjunto de resultados      3b
```

Métodos DatabaseMetaData para identificar el tipo de fuentes de datos

Puede utilizar los métodos DatabaseMetaData.getDatabaseProductName y DatabaseMetaData.getProductVersion para identificar el tipo y el nivel del gestor de bases de datos con el que está conectado y el sistema operativo en el que se está ejecutando el gestor de bases de datos.

DatabaseMetaData.getDatabaseProductName devuelve una serie que identifica el gestor de bases de datos y el sistema operativo. La serie tiene uno de estos formatos:

producto-basedatos
producto-basedatos/sistema-operativo

La tabla siguiente muestra ejemplos de valores devueltos por DatabaseMetaData.getDatabaseProductName.

Tabla 8. Ejemplos de valores de DatabaseMetaData.getDatabaseProductName

Valor de getDatabaseProductName	Producto de base de datos
DB2	DB2 para z/OS
DB2/LINUXX8664	DB2 Database para Linux, UNIX y Windows en Linux en x86
IBM Informix/UNIX64	IBM Informix en UNIX

DatabaseMetaData.getDatabaseVersionName devuelve una serie que contiene el indicador de producto de base de datos y el número de versión, número de release y nivel de mantenimiento de la fuente de datos.

La tabla siguiente muestra ejemplos de valores devueltos por DatabaseMetaData.getDatabaseProductVersion.

Tabla 9. Ejemplos de valores de DatabaseMetaData.getDatabaseProductVersion

Valor de getDatabaseProductVersion	Versión del producto de base de datos
DSN09015	DB2 para z/OS versión 9.1 en modalidad de función nueva
SQL09010	DB2 Database para Linux, UNIX y Windows versión 9.1
IFX11100	IBM Informix Versión 11.10

Extensiones de DatabaseMetaData para la obtención de información sobre los módulos

La clase com.ibm.db2.jcc.DB2DatabaseMetaData contiene métodos que permiten recuperar información sobre los procedimientos, las funciones y los tipos definidos por el usuario que se encuentran en módulos.

Un *módulo* es un objeto de base de datos formado por una recopilación de otros objetos de base de datos, como funciones, procedimientos, tipos y variables. Los módulos son similares a otras clases de Java. El objetivo principal de los módulos consiste en agrupar definiciones de objeto con un uso u objetivo empresarial común. Los módulos están soportados por DB2 Database para Linux, UNIX y Windows Versión 9.7 o posterior.

IBM Data Server Driver para JDBC y SQLJ ofrece en la interfaz DB2DatabaseMetaData los métodos siguientes para recuperar información sobre los objetos de base de datos que se encuentran en módulos. Cada método devuelve un conjunto de resultados que contiene todas las columnas en el método java.sql.DatabaseMetaData relacionado y una columna adicional que identifica el módulo en el que reside el objeto de base de datos.

Método	Método DatabaseMetaData relacionado	Información devuelta
getDBFunctionColumns	getFunctionColumns	Información sobre los parámetros y valores de retorno para las funciones definidas por el usuario o las funciones incorporadas que se encuentran en una fuente de datos.
getDBFunctions	getFunctions	Información sobre las funciones definidas por el usuario o las funciones incorporadas que se encuentran en una fuente de datos.
getDBProcedureColumns	getProcedureColumns	Información sobre los parámetros y valores de retorno para los procedimientos almacenados que se encuentran en una fuente de datos.

Método	Método DatabaseMetaData relacionado	Información devuelta
getDBProcedures	getProcedures	Información sobre los procedimientos almacenados que se encuentran en una fuente de datos.
getDBUDTs	getUDTs	Información sobre los tipos definidos por el usuario que se encuentran en una fuente de datos.

Si los mismos procedimientos almacenados, funciones o tipos definidos por el usuario están definidos en varios módulos distintos y se llama a uno de los métodos DatabaseMetaData, estos métodos devuelven más de una fila para el mismo procedimiento, función o tipo definido por el usuario. Estas filas son idénticas, salvo en la columna SPECIFICNAME. Para identificar de forma exclusiva los procedimientos almacenados, las funciones o los tipos definidos por el usuario por nombre de módulo, utilice los métodos DB2DatabaseMetaData.

Ejemplo

Supongamos que su ID es DB2ADMIN y crea dos módulos, llamados MYMOD1 y MYMOD2:

```
...
stmt.execute ("CREATE MODULE MYMOD1");
stmt.execute ("CREATE MODULE MYMOD2");
...
```

A continuación, crea un mismo procedimiento de SQL, llamado PROC1, en el módulo MYMOD1 y en el módulo MYMOD2:

```
...
stmt.execute ("ALTER MODULE MYMOD1 PUBLISH PROCEDURE PROC1 ( " +
                "IN PARM1 BOOLEAN, " +
                "INOUT PARM2 INTEGER) " +
                "LANGUAGE SQL " +
                "BEGIN...END");
stmt.execute ("ALTER MODULE MYMOD2 PUBLISH PROCEDURE PROC1 ( " +
                "IN PARM1 BOOLEAN, " +
                "INOUT PARM2 INTEGER) " +
                "LANGUAGE SQL " +
                "BEGIN...END");
```

El ejemplo siguiente utiliza el método DB2DatabaseMetaData.getDBProcedures para devolver información sobre todos los procedimientos llamados PROC1 en el esquema DB2ADMIN que se encuentran en módulos con un nombre del tipo "MYMOD%".

```
Connection con;
...                       // Establecer una conexión con la fuente de datos
DatabaseMetaData dbmd = con.getMetaData();
                        // Crear un objeto DatabaseMetaData
String schemaname="DB2ADMIN";
String modulename="MYMOD%";
String procname="PROC1";
                        // Indicar que se desea información
                        // sobre todos los procedimientos que cumplen estos
                        // criterios:
                        // Esquema: DB2ADMIN
                        // Módulo: MYMOD%
```

```
                        // Nombre de procedimiento: PROC1
ResultSet rs =
 ((com.ibm.db2.jcc.DB2DatabaseMetaData)dbmd).
   getDBProcedures(null,schemaname,modulename,procname);
                        // Convertir el objeto DatabaseMetaData
                        // en un objeto DB2DatabaseMetaData
                        // y llamar a DB2DatabaseMetaData.getDBProcedures
ResultSetMetaData rsmd = rs.getMetaData();
                        // Recuperar un objeto ResultSetMetaData del
                        // ResultSet que contiene la información sobre
                        // el procedimiento, para saber cuántas columnas
                        // debe recuperar y el nombre de cada columna.
while (rs.next()) {
  System.out.println ("----------- Row " + rowcount++ + " -----------");
                        // Recuperar cada fila del ResultSet
  for (int i = 1; i <= rsmd.getColumnCount (); i++) {
    System.out.println (i + ": " + rsmd.getColumnName (i) + " = " +
      rs.getString (i));
  }
}
```

Los valores que se devuelven son similares a los siguientes:

```
----------- Fila 1 -----------
1: PROCEDURE_CAT = null
2: PROCEDURE_SCHEM = DBADMIN
3: ROUTINENAME = PROC1
4: NUM_INPUT_PARAMS = 0
5: NUM_OUTPUT_PARAMS = 0
6: RESULT_SETS = 0
7: REMARKS = null
8: PROCEDURE_TYPE = 1
9: SPECIFICNAME = SQL090504144847100
10: MODULENAME = MYMOD1
----------- Fila 2 -----------
1: PROCEDURE_CAT = null
2: PROCEDURE_SCHEM = DBADMIN
3: ROUTINENAME = PROC1
4: NUM_INPUT_PARAMS = 0
5: NUM_OUTPUT_PARAMS = 0
6: RESULT_SETS = 0
7: REMARKS = null
8: PROCEDURE_TYPE = 1
9: SPECIFICNAME = SQL090504144847800
10: MODULENAME = MYMOD2
```

Variables en aplicaciones JDBC

Al igual que en cualquier otra aplicación Java, cuando escribe aplicaciones JDBC, debe declarar variables. En las aplicaciones Java, esas variables se conocen como identificadores Java.

Algunos de estos identificadores tienen la misma función que las variables de lenguaje principal tienen en otros lenguajes: contienen datos que se pasan a tablas de base de datos o que se reciben de ellas. En el código de programa siguiente, el identificador `empNo` contiene los datos recuperados de la columna de tabla EMPNO, cuyo tipo de datos es CHAR.

```
String empNo;
// Ejecutar una consulta y generar instancia del conjunto de resultados
rs = stmt.executeQuery("SELECT EMPNO FROM EMPLOYEE");
while (rs.next()) {
 String empNo = rs.getString(1);
 System.out.println("Número de empleado = " + empNo);
}
```

Su elección de tipos de datos Java puede afectar al rendimiento, pues DB2 selecciona mejores vías de acceso cuando los tipos de datos de las variables Java se corresponden estrechamente con los tipos de datos DB2.

Comentarios en una aplicación JDBC

Para documentar su programa JDBC, es necesario que incluya comentarios. Puede utilizar comentarios Java fuera de los métodos JDBC y comentarios Java o SQL en series de sentencia de SQL.

Puede incluir comentarios Java fuera de los métodos JDBC, siempre que el lenguaje Java lo permita. Dentro de una serie de sentencia de SQL en una llamada a método JDBC, puede utilizar comentarios en las ubicaciones siguientes:

- Para las conexiones con servidores de datos DB2 o servidores de datos Informix, los comentarios pueden ubicarse:
  - En cualquier punto de la serie de sentencia de SQL, especificados entre pares /* y */. Los pares /* y */ pueden anidarse.
  - Al final de la serie de sentencia de SQL, y precedidos por dos guiones (--).
- Para las conexiones con servidores de datos Informix únicamente, los comentarios pueden especificarse entre pares de llave izquierda ({) y llave derecha (}).

  Restricción: Un comentario especificado dentro de un par { y } no será válido si tiene aplicación cualquiera de las condiciones siguientes:
  - La serie de sentencia de SQL no es una llamada de procedimiento almacenado, la serie de sentencia de SQL va precedida y seguida de comentarios especificados entre pares { y } y el comentario del inicio de la serie de la sentencia de SQL empieza con la palabra `call`.
  - La serie de sentencia de SQL es una llamada de procedimiento almacenado y el comentario {call} se encuentra al inicio de la sintaxis de escape para la llamada de procedimiento almacenado.
  - El comentario contiene cualquiera de los caracteres siguientes:
    - Comilla simple (')
    - Comilla doble (")
    - Llave izquierda ({)
    - Llave derecha (})
    - /*
  - El comentario puede interpretarse como sintaxis de escape de SQL. Los comentarios que empiezan con los caracteres siguientes pueden interpretarse como sintaxis de escape de SQL:
    - d seguida de un espacio
    - t seguida de un espacio
    - ts seguidas de un espacio
    - escape seguido de un espacio
    - oj seguidas de un espacio
    - fn seguidas de un espacio

  Por ejemplo, no son válidas las siguientes series de sentencia de SQL:

```
"{call comment at beginning} select * from systables {ending comment}"
"{{call} call mysp(?, ?)}"
```

Interfaces JDBC para ejecutar SQL

Puede ejecutar sentencias de SQL dentro de un programa de SQL estándar para actualizar datos de tablas, recuperar datos de tablas o invocar procedimientos almacenados. Para ejecutar las mismas funciones en un programa JDBC, debe invocar métodos.

Esos métodos están definidos en las interfaces siguientes:

- La interfaz Statement soporta la ejecución de todas las sentencias de SQL. Las interfaces siguientes heredan métodos de la interfaz Statement:
  - La interfaz PreparedStatement soporta cualquier sentencia de SQL que contenga marcadores de parámetros de entrada. Los marcadores de parámetros representan variables de entrada. La interfaz PreparedStatement también se puede utilizar para sentencias de SQL sin marcadores de parámetros.

    Con IBM Data Server Driver para JDBC y SQLJ, la interfaz PreparedStatement permite invocar procedimientos almacenados que tienen parámetros de entrada y ningún parámetro de salida, y que no devuelven ningún conjunto de resultados. Sin embargo, la interfaz preferida es CallableStatement.
  - La interfaz CallableStatement soporta la invocación de un procedimiento almacenado.

    La interfaz CallableStatement permite invocar procedimientos almacenados con parámetros de entrada, parámetros de salida, con ambas clases de parámetros o sin parámetros. Con IBM Data Server Driver para JDBC y SQLJ, también se puede utilizar la interfaz Statement para llamar a procedimientos almacenados, pero éstos no deben tener parámetros.
- La interfaz ResultSet proporciona acceso a los resultados generados por una consulta. La interfaz ResultSet tiene la misma finalidad que el cursor utilizado en las aplicaciones de SQL para otros lenguajes de programación.

Creación y modificación de objetos de base de datos utilizando el método Statement.executeUpdate

El método Statement.executeUpdate es uno de los métodos JDBC que puede utilizar para actualizar tablas e invocar procedimientos almacenados.

Acerca de esta tarea

Puede utilizar el método Statement.executeUpdate para realizar las acciones siguientes:

- Ejecutar sentencias de definición de datos, tales como CREATE, ALTER, DROP, GRANT y REVOKE
- Ejecutar sentencias INSERT, UPDATE, DELETE y MERGE que no contienen marcadores de parámetros.
- Con IBM Data Server Driver para JDBC y SQLJ, ejecutar la sentencia CALL para invocar procedimientos almacenados que carecen de parámetros y no devuelven conjuntos de resultados.

Para ejecutar esas sentencias de SQL, debe seguir estos pasos:

Procedimiento

1. Invoque el método Connection.createStatement para crear un objeto Statement.
2. Invoque el método Statement.executeUpdate para ejecutar la operación de SQL.
3. Invoque el método Statement.close para cerrar el objeto Statement.

Ejemplo

Suponga que desea ejecutar esta sentencia de SQL:

```
UPDATE EMPLOYEE SET PHONENO='4657' WHERE EMPNO='000010'
```

El código siguiente crea el objeto Statement denominado `stmt`, ejecuta la sentencia UPDATE y devuelve en `numUpd` el número de filas que fueron actualizadas. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
Connection con;
Statement stmt;
int numUpd;
...
stmt = con.createStatement();               // Crear un objeto Statement   1
numUpd = stmt.executeUpdate(
  "UPDATE EMPLOYEE SET PHONENO='4657' WHERE EMPNO='000010'");               2
                                            // Ejecutar la actualización
stmt.close();                               // Cerrar el objeto Statement  3
```

Figura 8. Utilización de Statement.executeUpdate

Actualización de datos de tablas utilizando el método PreparedStatement.executeUpdate

El método Statement.executeUpdate es efectivo si actualiza tablas DB2 con valores constantes. Sin embargo, las actualizaciones a menudo suponen pasar a tablas DB2 valores contenidos en variables. Para hacer esto, utilice el método PreparedStatement.executeUpdate.

Acerca de esta tarea

Con IBM Data Server Driver para JDBC y SQLJ, puede también utilizar PreparedStatement.executeUpdate para invocar procedimientos almacenados que tienen parámetros de entrada y ningún parámetro de salida, y que no devuelven ningún conjunto de resultados.

DB2 para z/OS no es compatible con la ejecución dinámica de la sentencia CALL. Para las llamadas a procedimientos almacenados que residen en fuentes de datos DB2 para z/OS, los parámetros pueden ser marcadores de parámetros o literales, pero no expresiones. Se pueden utilizar los tipos de literales siguientes:

- Entero
- Double
- Decimal
- Carácter
- Hexadecimal
- Graphic

Para las llamadas a procedimientos almacenados que residen en fuentes de datos IBM Informix, el objeto PreparedStatement puede ser una sentencia CALL o una sentencia EXECUTE PROCEDURE.

Cuando ejecuta una sentencia de SQL muchas veces, puede obtener un mejor rendimiento creando la sentencia de SQL en forma de objeto PreparedStatement.

Por ejemplo, la siguiente sentencia UPDATE permite actualizar la tabla de empleados (EMPLOYEE) solamente para un único número de teléfono y número de empleado:

```
UPDATE EMPLOYEE SET PHONENO='4657' WHERE EMPNO='000010'
```

Suponga que desea generalizar la operación para poder actualizar la tabla de empleados para un conjunto cualquiera de números de teléfono y números de empleado. Para ello es necesario que sustituya los valores constantes del número de teléfono y número de empleado por variables:

```
UPDATE EMPLOYEE SET PHONENO=? WHERE EMPNO=?
```

Las variables de esta clase se denominan marcadores de parámetros. Para ejecutar una sentencia de SQL con marcadores de parámetros, debe seguir estos pasos:

Procedimiento

1. Invoque el método Connection.prepareStatement para crear un objeto PreparedStatement.
2. Invoque los métodos PreparedStatement.setXXX para pasar valores a las variables de entrada.

   Este paso presupone que está utilizando marcadores de parámetro estándar. Alternativamente, si utiliza marcadores de parámetro con nombre, utilice los métodos exclusivos de IBM Data Server Driver para JDBC y SQLJ para pasar valores a los parámetros de entrada.
3. Invoque el método PreparedStatement.executeUpdate para actualizar la tabla con los valores variables.
4. Invoque el método PreparedStatement.close para cerrar el objeto PreparedStatement cuando termine de utilizar ese objeto.

Ejemplo

El código siguiente ejecuta los pasos anteriores para actualizar el número de teléfono '4657' del empleado cuyo número de empleado es '000010'. Los números situados a la derecha de determinadas sentencias corresponden a pasos descritos anteriormente.

```
Connection con;
PreparedStatement pstmt;
int numUpd;
...
pstmt = con.prepareStatement(
  "UPDATE EMPLOYEE SET PHONENO=? WHERE EMPNO=?");
                                // Crear un objeto PreparedStatement       1
pstmt.setString(1,"4657");      // Asignar primer valor a primer parámetro 2
pstmt.setString(2,"000010");    // Asignar primer valor a segundo parámetro
numUpd = pstmt.executeUpdate(); // Realizar primera actualización          3
pstmt.setString(1,"4658");      // Asignar segundo valor a primer parámetro
pstmt.setString(2,"000020");    // Asignar segundo valor a segundo parámetro
numUpd = pstmt.executeUpdate(); // Ejecutar segunda actualización
pstmt.close();                  // Cerrar el objeto PreparedStatement      4
```

Figura 9. Uso de PreparedStatement.executeUpdate para una sentencia de SQL con marcadores de parámetros

Puede también utilizar el método PreparedStatement.executeUpdate para sentencias que no tienen marcadores de parámetros. Los pasos para ejecutar un objeto PreparedStatement sin marcadores de parámetros son similares a los pasos para ejecutar el objeto PreparedStatement con marcadores de parámetros, excepto que se omite el paso 2. El ejemplo siguiente muestra estos pasos.

```
Connection con;
PreparedStatement pstmt;
int numUpd;
...
pstmt = con.prepareStatement(
  "UPDATE EMPLOYEE SET PHONENO='4657' WHERE EMPNO='000010'");
                                  // Crear un objeto PreparedStatement      1
numUpd = pstmt.executeUpdate();   // Realizar la actualización              3
pstmt.close();                    // Cerrar el objeto PreparedStatement     4
```

Figura 10. Uso de PreparedStatement.executeUpdate para una sentencia de SQL sin marcadores de parámetros

Métodos executeUpdate de JDBC sobre un servidor DB2 para z/OS

El estándar JDBC establece que el método executeUpdate devuelve un número de filas o el valor 0. Pero si el método executeUpdate se ejecuta sobre un servidor DB2 para z/OS, ese método puede devolver el valor -1.

Para las sentencias executeUpdate emitidas sobre un servidor DB2 para z/OS, el valor devuelto depende del tipo de sentencia de SQL que se está ejecutando:

- Para una sentencia de SQL que pueda tener un recuento de actualizaciones, como una sentencia INSERT, UPDATE, DELETE o MERGE, el valor que se devuelve es el número de filas afectadas. Puede ser:
  - Un número positivo, si la operación afecta a un número positivo de filas y la operación no es una supresión masiva sobre un espacio de tablas segmentado.
  - 0, si la operación no afecta a ninguna fila.
  - -1, si la operación es una supresión masiva sobre un espacio de tablas segmentado.
- Para una sentencia CALL de SQL, el valor devuelto es -1, pues la fuente de datos no puede determinar el número de filas afectadas. Las llamadas a getUpdateCount o getMoreResults correspondientes a una sentencia CALL también devuelven -1.
- Para cualquier otra sentencia de SQL, se devuelve un valor de -1.

Realización de actualizaciones por lotes en aplicaciones JDBC

En las actualizaciones de proceso por lotes, en lugar de actualizar filas de una tabla de forma individual, puede hacer que JDBC ejecute un grupo de actualizaciones al mismo tiempo. Las sentencias que se pueden incluir en un mismo lote de actualizaciones se denominan sentencias *procesables por lotes*.

Acerca de esta tarea

Si una sentencia tiene parámetros de entrada o expresiones de lenguaje principal, puede incluir esa sentencia solo en un lote que tenga otras instancias de la misma sentencia. Este tipo de lote se denomina *lote homogéneo*. Si una sentencia carece de parámetros de entrada, puede incluir esa sentencia en un lote solo si las demás sentencias del lote no tienen parámetros de entrada ni expresiones de lenguaje principal. Este tipo de lote se denomina *lote heterogéneo*. Dos sentencias que se puedan incluir en el mismo lote se dice que son *compatibles por lote*.

Utilice los siguientes métodos de Statement para crear, ejecutar y eliminar un lote de actualizaciones SQL:
- addBatch
- executeBatch
- clearBatch

Utilice el siguiente método de PreparedStatement y CallableStatement para crear un lote de parámetros para que una sentencia individual se pueda ejecutar varias veces en un lote, con un conjunto de parámetros diferente para cada ejecución.
- addBatch

Restricciones en la ejecución de las sentencias de un lote:

- Si intenta ejecutar una sentencia SELECT en un lote, se emite una excepción BatchUpdateException.
- Un objeto CallableStatement que ejecute en un lote puede contener parámetros de salida. Sin embargo, no puede recuperar los valores de los parámetros de salida. Si intenta hacerlo, se emite una excepción BatchUpdateException.
- No puede recuperar objetos ResultSet de un objeto CallableStatement que ejecute en un lote. En ese caso no se emite una excepción BatchUpdateException, pero la invocación del método getResultSet devuelve un valor nulo.

Para realizar actualizaciones por lotes utilizando varias sentencias sin parámetros de entrada, siga estos pasos básicos:
1. Para cada sentencia de SQL que desee ejecutar en el lote, invoque el método addBatch.
2. Invoque el método executeBatch para ejecutar el lote de sentencias.
3. Compruebe si se han producido errores. Si no han ocurrido errores:
   a. Obtenga el número de filas afectadas por cada sentencia de SQL a partir de la matriz devuelta por la invocación de executeBatch. Este número no incluye las filas afectadas por activadores o por la aplicación de la integridad referencial.
   b. Si `AutoCommit` está inhabilitado para el objeto Connection, invoque el método commit para confirmar los cambios.

      Si `AutoCommit` está habilitado para el objeto Connection, IBM Data Server Driver para JDBC y SQLJ añade un método commit al final del proceso por lotes.

Para realizar actualizaciones por lotes utilizando una sola sentencia con varios conjuntos de parámetros de entrada, siga estos pasos básicos:

Procedimiento

1. Si la sentencia de proceso por lotes es una sentencia UPDATE buscada, una sentencia MERGE o una sentencia DELETE buscada, establezca la modalidad de confirmación automática para la conexión en `false`.
2. Invoque el método prepareStatement para crear un objeto PreparedStatement.
3. Para cada conjunto de valores de parámetros de entrada:
   a. Ejecute métodos setXXX para asignar valores a los parámetros de entrada.
   b. Invoque el método addBatch para añadir el conjunto de parámetros de entrada al lote.
4. Invoque el método executeBatch para ejecutar las sentencias con todos los conjuntos de parámetros.
5. Si no han ocurrido errores:

a. Obtenga el número de filas afectadas por cada ejecución de la sentencia de SQL a partir de la matriz devuelta por la invocación de executeBatch. El número de filas afectadas no incluye las filas afectadas por activadores o por la aplicación de la integridad referencial.

   Si se cumplen las condiciones siguientes, IBM Data Server Driver para JDBC y SQLJ devuelve `Statement.SUCCESS_NO_INFO` (-2), en lugar del número de filas afectadas por cada sentencia de SQL:

   - La aplicación está conectada con un subsistema que está en DB2 para z/OS Versión 8 en modalidad de función nueva, o posteriores.
   - La aplicación utiliza la Versión 3.1 o posteriores de IBM Data Server Driver para JDBC y SQLJ.
   - IBM Data Server Driver para JDBC y SQLJ utiliza operaciones INSERT de varias filas para ejecutar las actualizaciones por lotes.

   Esto se debe a que, al realizar una operación INSERT de varias filas, el servidor de bases de datos ejecuta todo el proceso por lotes como una única operación, de forma que no devuelve resultados para sentencias de SQL individuales.

b. Si `AutoCommit` está inhabilitado para el objeto Connection, invoque el método commit para confirmar los cambios.

   Si `AutoCommit` está habilitado para el objeto Connection, IBM Data Server Driver para JDBC y SQLJ añade un método commit al final del proceso por lotes.

c. Si el objeto PreparedStatement devuelve claves generadas automáticamente, invoque DB2PreparedStatement.getDBGeneratedKeys para recuperar una matriz de objetos ResultSet que contiene las claves generadas automáticamente.

   Compruebe la longitud de la matriz devuelta. Si la longitud de la matriz devuelta es 0, se ha producido un error durante la recuperación de las claves generadas automáticamente.

6. Si se han producido errores, procese BatchUpdateException.

Ejemplo

En el siguiente fragmento de código de programa, se procesan por lotes dos conjuntos de parámetros. Luego, una sentencia UPDATE que admite dos parámetros de entrada se ejecuta dos veces, una vez con cada conjunto de parámetros. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
try {
...
  PreparedStatement preps = conn.prepareStatement(
    "UPDATE DEPT SET MGRNO=? WHERE DEPTNO=?");          2
  ps.setString(1,mgrnum1);                              3a
  ps.setString(2,deptnum1);
  ps.addBatch();                                        3b

  ps.setString(1,mgrnum2);
  ps.setString(2,deptnum2);
  ps.addBatch();
  int [] numUpdates=ps.executeBatch();                  4
  for (int i=0; i < numUpdates.length; i++) {           5a
    if (numUpdates[i] == SUCCESS_NO_INFO)
      System.out.println("Execution " + i +
        ": unknown number of rows updated");
    else
      System.out.println("Execution " + i +
```

```
        "successful: " numUpdates[i] + " rows updated");
  }
  conn.commit();                                              5b
} catch(BatchUpdateException b) {                             6
  // process BatchUpdateException
}
```

En el código de programa siguiente, una sentencia INSERT de proceso por lotes devuelve claves generadas automáticamente.

```
import java.sql.*;
import com.ibm.db2.jcc.*;
...
Connection conn;
...
try {
...
  PreparedStatement ps = conn.prepareStatement(               2
    "INSERT INTO DEPT (DEPTNO, DEPTNAME, ADMRDEPT) " +
    "VALUES (?,?,?)",
    Statement.RETURN_GENERATED_KEYS);
  ps.setString(1,"X01");                                      3a
  ps.setString(2,"Finance");
  ps.setString(3,"A00");
  ps.addBatch();                                              3b
  ps.setString(1,"Y01");
  ps.setString(2,"Accounting");
  ps.setString(3,"A00");
  ps.addBatch();

  int [] numUpdates=preps.executeBatch();                     4

  for (int i=0; i < numUpdates.length; i++) {                 5a
    if (numUpdates[i] == SUCCESS_NO_INFO)
      System.out.println("Execution " + i +
        ": unknown number of rows updated");
    else
      System.out.println("Execution " + i +
        "successful: " numUpdates[i] + " rows updated");
  }
  conn.commit();                                              5b
  ResultSet[] resultList =
    ((DB2PreparedStatement)ps).getDBGeneratedKeys();          5c
  if (resultList.length != 0) {
    for (i = 0; i < resultList.length; i++) {
        while (resultList[i].next()) {
          java.math.BigDecimal idColVar = rs.getBigDecimal(1);
                              // Obtener valor de clave generada
                              // automáticamente
          System.out.println("Valor de clave generada automáticamente = "
            + idColVar);
        }
    }
  }
  else {
    System.out.println("Error al recuperar claves generadas automáticamente");
  }
} catch(BatchUpdateException b) {                             6
  // process BatchUpdateException
}
```

En el código de programa siguiente, una sentencia UPDATE de proceso por lotes devuelve claves generadas automáticamente. El código da nombre a la columna DEPTNO como una clave generada automáticamente, actualiza dos filas en la tabla DEPT de un proceso por lotes y recupera los valores de DEPTNO para las filas

actualizadas. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
import java.sql.*;
import com.ibm.db2.jcc.*;
...
Connection conn;
...
String[] agkNames = {"DEPTNO"};
try {
...
  conn.setAutoCommit(false);                                    1
  PreparedStatement ps = conn.prepareStatement(                 2
    "UPDATE DEPT SET  DEPTNAME=? " +
    "WHERE DEPTNO=?",agkNames);
  ps.setString(1,"X01");                                        3a
  ps.setString(2,"Planning");
  ps.addBatch();                                                3b
  ps.setString(1,"Y01");
  ps.setString(2,"Bookkeeping");
  ps.addBatch();

  int [] numUpdates=ps.executeBatch();                          4

  for (int i=0; i < numUpdates.length; i++) {                   5a
    if (numUpdates[i] == SUCCESS_NO_INFO)
      System.out.println("Execution " + i +
        ": unknown number of rows updated");
    else
      System.out.println("Execution " + i +
        "successful: " numUpdates[i] + " rows updated");
  }
  conn.commit();                                                5b
  ResultSet[] resultList =
    ((DB2PreparedStatement)ps).getDBGeneratedKeys();            5c
  if (resultList.length != 0) {
    for (i = 0; i < resultList.length; i++) {
        while (resultList[i].next()) {
          String deptNoKey = rs.getString(1);
                             // Obtener valor de clave generada
                             // automáticamente
          System.out.println("Valor de clave generada automáticamente = "
            + deptNoKey);
        }
    }
  }
  else {
    System.out.println("Error al recuperar claves generadas automáticamente");
  }
}
catch(BatchUpdateException b) {                                 6
  // process BatchUpdateException
}
```

Obtención de información acerca de parámetros de PreparedStatement mediante métodos ParameterMetaData

IBM Data Server Driver para JDBC y SQLJ incluye soporte para la interfaz ParameterMetaData. La interfaz ParameterMetaData contiene métodos obtienen información sobre los marcadores de parámetros de un objeto PreparedStatement.

Acerca de esta tarea

Los métodos de ParameterMetaData proporcionan los tipos de información siguientes:

- Los tipos de datos de los parámetros, incluida la precisión y escala de los parámetros decimales.
- Los nombres de tipos de los parámetros, específicos de la base de datos. Para los parámetros que corresponden a columnas de tabla que están definidas con tipos diferenciados, estos nombres son los nombres de los tipos diferenciados.
- Indicación de si los parámetros pueden contener nulos.
- Indicación de si los parámetros son parámetros de entrada o de salida.
- Indicación de si los valores de un parámetro numérico pueden tener signo.
- El nombre de clase Java totalmente calificado que el objeto PreparedStatement.setObject utiliza cuando define un valor de parámetro.

Para invocar métodos de ParameterMetaData, debe seguir estos pasos básicos:

Procedimiento

1. Invoque el método Connection.prepareStatement para crear un objeto PreparedStatement.
2. Invoque el método PreparedStatement.getParameterMetaData para obtener un objeto ParameterMetaData.
3. Invoque ParameterMetaData.getParameterCount para determinar el número de parámetros de PreparedStatement.
4. Invoque métodos de ParameterMetaData para parámetros individuales.

Ejemplo

El código de programa siguiente muestra cómo utilizar métodos ParameterMetaData para determinar el número y los tipos de datos de los parámetros de una sentencia UPDATE de SQL. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
Connection con;
ParameterMetaData pmtadta;
int mtadtacnt;
String sqlType;
...
pstmt = con.prepareStatement(
  "UPDATE EMPLOYEE SET PHONENO=? WHERE EMPNO=?");
                                  // Crear un objeto PreparedStatement      1
pmtadta = pstmt.getParameterMetaData();                                     2
                                  // Crear un objeto ParameterMetaData
mtadtacnt = pmtadta.getParameterCount();                                    3
                                  // Determinar el número de parámetros
System.out.println("Número de parámetros de sentencia: " + mtadtacnt);
for (int i = 1; i <= mtadtacnt; i++) {
  sqlType = pmtadta.getParameterTypeName(i);                                4
                                  // Obtener tipo de datos de SQL para
                                  // cada parámetro
 System.out.println("Tipo de SQL del parámetro " + i " es " + sqlType);
}
...
pstmt.close();                    // Cerrar PreparedStatement
```

Figura 11. Uso de métodos de ParameterMetaData para obtener información sobre un objeto PreparedStatement

Recuperación de datos en aplicaciones JDBC

Utilice objetos ResultSet para recuperar datos en las aplicaciones JDBC. Un ResultSet representa el conjunto de resultados de una consulta.

Recuperación de datos de tablas utilizando el método Statement.executeQuery

Para recuperar datos de una tabla utilizando una sentencia SELECT sin marcadores de parámetros, puede utilizar el método Statement.executeQuery.

Acerca de esta tarea

Este método devuelve una tabla de resultados en un objeto ResultSet. Una vez obtenida la tabla de resultados, debe utilizar métodos de ResultSet para desplazarse por la tabla de resultados y obtener los valores individuales de cada columna de cada fila.

Con IBM Data Server Driver para JDBC y SQLJ, también puede utilizar el método Statement.executeQuery para obtener un conjunto de resultados de una llamada de procedimiento almacenado, si ese procedimiento almacenado devuelve un solo conjunto de resultados. Si el procedimiento almacenado devuelve varios conjuntos de resultados, debe utilizar el método Statement.execute.

Este tema describe la modalidad más sencilla de ResultSet, que es un objeto ResultSet de solo lectura en el que el usuario solo puede desplazarse hacia delante, una fila cada vez. IBM Data Server Driver para JDBC y SQLJ también permite utilizar ResultSet actualizables y desplazables.

Para recuperar filas de una tabla utilizando una sentencia SELECT sin marcadores de parámetros, siga estos pasos:

Procedimiento

1. Invoque el método Connection.createStatement para crear un objeto Statement.
2. Invoque el método Statement.executeQuery para obtener la tabla de resultados de la sentencia SELECT en un objeto ResultSet.
3. En un bucle, posicione el cursor utilizando el método next y recupere datos de cada columna de la fila actual del objeto ResultSet utilizando métodos getXXX. `XXX` representa un tipo de datos.
4. Invoque el método ResultSet.close para cerrar el objeto ResultSet.
5. Invoque el método Statement.close para cerrar el objeto Statement cuando termine de utilizar ese objeto.

Ejemplo

El código de programa siguiente muestra cómo recuperar todas las filas de la tabla EMPLOYEE. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
String empNo;
Connection con;
Statement stmt;
ResultSet rs;
...
stmt = con.createStatement();     // Crear un objeto Statement          1
rs = stmt.executeQuery("SELECT EMPNO FROM EMPLOYEE");                   2
                                  // Obtener tabla de resultados de la consulta
while (rs.next()) {               // Situar el cursor                   3
 empNo = rs.getString(1);         // Obtener solo el valor de la primera columna
 System.out.println("Número de empleado = " + empNo);
                                  // Imprimir el valor de columna
}
rs.close();                       // Cerrar el conjunto de resultados   4
stmt.close();                     // Cerrar la sentencia                5
```

Figura 12. Utilización de Statement.executeQuery

Recuperación de datos de tablas utilizando el método PreparedStatement.executeQuery

Para obtener datos de una tabla utilizando una sentencia SELECT con marcadores de parámetros, utilice el método PreparedStatement.executeQuery.

Acerca de esta tarea

Este método devuelve una tabla de resultados en un objeto ResultSet. Una vez obtenida la tabla de resultados, debe utilizar métodos de ResultSet para desplazarse por la tabla de resultados y obtener los valores individuales de cada columna de cada fila.

Con IBM Data Server Driver para JDBC y SQLJ, también puede utilizar el método PreparedStatement.executeQuery para obtener un conjunto de resultados de una llamada de procedimiento almacenado, si ese procedimiento almacenado devuelve un solo conjunto de resultados y tiene solamente parámetros de entrada. Si el procedimiento almacenado devuelve varios conjuntos de resultados, debe utilizar el método PreparedStatement.execute.

Puede también utilizar el método PreparedStatement.executeQuery para sentencias que no tienen marcadores de parámetros. Cuando ejecuta una consulta muchas veces, puede obtener un mejor rendimiento creando la sentencia de SQL en forma de objeto PreparedStatement.

Para obtener filas de una tabla utilizando una sentencia SELECT con marcadores de parámetros, siga estos pasos:

Procedimiento

1. Invoque el método Connection.prepareStatement para crear un objeto PreparedStatement.
2. Invoque métodos PreparedStatement.setXXX para pasar valores a los parámetros de entrada.
3. Invoque el método PreparedStatement.executeQuery para obtener la tabla de resultados de la sentencia SELECT en un objeto ResultSet.

   Restricción: Para una PreparedStatement que contiene un predicado IN, la expresión que es el argumento del predicado IN no puede tener más de 32767 parámetros si el servidor de datos de destino es un sistema DB2 Database para

Linux, UNIX y Windows. De lo contrario, IBM Data Server Driver para JDBC y SQLJ emite una SQLException con el código de error -4499.

4. En un bucle, posicione el cursor utilizando el método ResultSet.next y recupere datos de cada columna de la fila actual del objeto ResultSet utilizando métodos getXXX.
5. Invoque el método ResultSet.close para cerrar el objeto ResultSet.
6. Invoque el método PreparedStatement.close para cerrar el objeto PreparedStatement cuando termine de utilizar ese objeto.

Ejemplo

El código de programa siguiente muestra cómo recuperar filas de la tabla EMPLOYEE para un empleado determinado. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
String empnum, phonenum;
Connection con;
PreparedStatement pstmt;
ResultSet rs;
...
pstmt = con.prepareStatement(
  "SELECT EMPNO, PHONENO FROM EMPLOYEE WHERE EMPNO=?");
                                  // Crear un objeto PreparedStatement      1
pstmt.setString(1,"000010");      // Asignar valor a parámetro de entrada   2

rs = pstmt.executeQuery();        // Obtener tabla resultados de consulta   3
while (rs.next()) {               // Situar el cursor                       4
 empnum = rs.getString(1);    // Obtener el valor de la primera columna
 phonenum = rs.getString(2);  // Obtener el valor de la primera columna
 System.out.println("Número de empleado = " + empnum +
   "Número de teléfono = " + phonenum);
                                  // Imprimir valores de columnas
}
rs.close();                       // Cerrar el conjunto de resultados       5
pstmt.close();                    // Cerrar PreparedStatement               6
```

Figura 13. Ejemplo de utilización de PreparedStatement.executeQuery

Realización de consultas por lotes en aplicaciones JDBC

IBM Data Server Driver para JDBC y SQLJ proporciona una interfaz DB2PreparedStatement específica de IBM Data Server Driver para JDBC y SQLJ que le permite realizar consultas de proceso por lotes en un lote homogéneo.

Acerca de esta tarea

Para realizar consultas por lotes utilizando una sola sentencia con varios conjuntos de parámetros de entrada, siga estos pasos básicos:

Procedimiento

1. Invoque el método prepareStatement para crear un objeto PreparedStatement para la sentencia de SQL con parámetros de entrada.
2. Para cada conjunto de valores de parámetros de entrada:
   a. Ejecute los métodos PreparedStatement.setXXX para asignar valores a los parámetros de entrada.
   b. Invoque el método PreparedStatement.addBatch para añadir el conjunto de parámetros de entrada al lote.

3. Convierta el objeto PreparedStatement en un objeto DB2PreparedStatement e invoque el método DB2PreparedStatement.executeDB2QueryBatch para ejecutar la sentencia con todos los conjuntos de parámetros.
4. En un bucle, recupere los objetos ResultSet:
   a. Recupere cada objeto ResultSet.
   b. Recupere todas las filas de cada objeto ResultSet.

Ejemplo

Ejemplo: En el siguiente fragmento de código de programa, se procesan por lotes dos conjuntos de parámetros. Luego, una sentencia SELECT, que admite un parámetro de entrada, se ejecuta dos veces, una vez con cada valor de parámetro. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
java.sql.Connection con = java.sql.DriverManager.getConnection(url, properties);
java.sql.Statement s = con.createStatement();
// Limpiar desde las ejecuciones anteriores
try {
  s.executeUpdate ("drop table TestQBatch");
}
catch (Exception e) {
}

// Crear y llenar una tabla de prueba
s.executeUpdate ("create table TestQBatch (col1 int, col2 char(10))");
s.executeUpdate ("insert into TestQBatch values (1, 'test1')");
s.executeUpdate ("insert into TestQBatch values (2, 'test2')");
s.executeUpdate ("insert into TestQBatch values (3, 'test3')");
s.executeUpdate ("insert into TestQBatch values (4, 'test4')");
s.executeUpdate ("insert into TestQBatch values (1, 'test5')");
s.executeUpdate ("insert into TestQBatch values (2, 'test6')");

try {
  PreparedStatement pstmt =                                    1
    con.prepareStatement("Select * from TestQBatch where col1 = ?");
  pstmt.setInt(1,1);                                           2a
  pstmt.addBatch();                                            2b
  // Añadir algunos valores más al lote
  pstmt.setInt(1,2);
  pstmt.addBatch();
  pstmt.setInt(1,3);
  pstmt.addBatch();
  pstmt.setInt(1,4);
  pstmt.addBatch();
  ((com.ibm.db2.jcc.DB2PreparedStatement)pstmt).executeDB2QueryBatch();
                                                               3
} catch(BatchUpdateException b) {
  // process BatchUpdateException
}
ResultSet rs;
while(pstmt.getMoreResults()) {                                4
  rs = pstmt.getResultSet();                                   4a
  while (rs.next()) {                                          4b
    System.out.print (rs.getInt (1) + "  ");
    System.out.println (rs.getString (2));
  }
  System.out.println();
  rs.close ();
}
// Clean up
s.close ();
pstmt.close();
con.close ();
```

Obtención de información acerca de un conjunto de resultados utilizando métodos ResultSetMetaData

No se puede conocer siempre el número de columnas y tipos de datos de las columnas de una tabla o conjunto de resultados. Esto es especialmente cierto cuando está recuperando datos de una fuente de datos remota.

Acerca de esta tarea

Cuando escriba programas que obtienen ResultSet desconocidos, es necesario utilizar métodos de ResultSetMetaData para determinar las características de los ResultSet antes de poder recuperar datos de ellos.

Los métodos de ResultSetMetaData proporcionan los tipos de información siguientes:

- El número de columnas de un ResultSet
- El calificador de la tabla subyacente del ResultSet
- Información sobre una columna, tal como el tipo de datos, la longitud, la precisión, la escala y la posibilidad de contener nulos
- La indicación de si una columna es de solo lectura

Después de invocar el método executeQuery para generar el ResultSet de una consulta sobre una tabla, siga estos pasos básicos para determinar el contenido del ResultSet:

Procedimiento

1. Invoque el método getMetaData para el objeto ResultSet para crear un objeto ResultSetMetaData.
2. Invoque el método getColumnCount para determinar el número de columnas del ResultSet.
3. Para cada columna del ResultSet, ejecute métodos de ResultSetMetaData para determinar las características de las columnas.

   Los resultados de la llamada ResultSetMetaData.getColumnName reflejan la información sobre el nombre de la columna y que está almacenada en el catálogo de DB2 de dicha fuente de datos.

Ejemplo

El código de programa siguiente muestra cómo determinar los tipos de datos de todas las columnas de la tabla EMPLOYEE. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
String s;
Connection con;
Statement stmt;
ResultSet rs;
ResultSetMetaData rsmtadta;
int colCount
int mtadtaint;
int i;
String colName;
String colType;
...
stmt = con.createStatement();         // Crear un objeto Statement
rs = stmt.executeQuery("SELECT * FROM EMPLOYEE");
                                      // Obtener conjunto de resultados de la consulta
rsmtadta = rs.getMetaData();     // Crear un objeto ResultSetMetaData   1
colCount = rsmtadta.getColumnCount();                                  2
                                      // Encontrar número de columnas de EMP
for (i=1; i<= colCount; i++) {                                         3
 colName = rsmtadta.getColumnName();    // Obtener nombre de columna
 colType = rsmtadta.getColumnTypeName();
                                          // Obtener tipo de datos de la columna
 System.out.println("Columna = " + colName +
  " es del tipo de datos " + colType);
                                          // Imprimir el valor de columna
}
```

Figura 14. Uso de métodos de ResultSetMetaData para obtener información sobre un conjunto de resultados

Características de un conjunto de resultados de JDBC cuando se utiliza IBM Data Server Driver para JDBC y SQLJ

IBM Data Server Driver para JDBC y SQLJ es compatible con cursores desplazables, actualizables y con capacidad de retención.

Además de avanzar fila a fila por un ResultSet, puede ser deseable poder hacer lo siguiente:

- Retroceder o ir directamente a una fila específica
- Actualizar, suprimir o insertar filas en un ResultSet
- Dejar abierto el ResultSet después de una operación COMMIT

Los términos siguientes describen características de un ResultSet:

capacidad de desplazamiento
Capacidad del cursor del ResultSet para avanzar solamente, o avanzar una o más filas, retroceder una o más filas o ir hasta una fila determinada.

Si un cursor de un ResultSet es desplazable, también tiene un atributo de sensibilidad, que describe si el cursor es sensible a los cambios producidos en la tabla subyacente.

capacidad de actualización
Capacidad de poder utilizar el cursor para actualizar o suprimir filas. Esta característica no es aplicable a un ResultSet devuelto por un procedimiento almacenado, pues un ResultSet de un procedimiento almacenado no se puede actualizar.

capacidad de retención
Capacidad del cursor para permanecer abierto después de una operación COMMIT.

Las características de un ResultSet correspondientes a la capacidad de actualización, capacidad de desplazamiento y capacidad de retención se definen mediante parámetros en los métodos Connection.prepareStatement o `Connection.createStatement`. Los valores de un ResultSet se corresponden con atributos de un cursor en la base de datos. La tabla siguiente muestra los valores de capacidad de desplazamiento, capacidad de actualización y capacidad de retención en JDBC y los correspondientes atributos de cursor.

Tabla 10. Características de un conjunto de resultados en JDBC y los atributos de cursor en SQL

Valor de JDBC	Valor de cursor de DB2	Valor de cursor de IBM Informix
CONCUR_READ_ONLY	FOR READ ONLY	FOR READ ONLY
CONCUR_UPDATABLE	FOR UPDATE	FOR UPDATE
HOLD_CURSORS_OVER_COMMIT	WITH HOLD	WITH HOLD
TYPE_FORWARD_ONLY	SCROLL no especificado	SCROLL no especificado
TYPE_SCROLL_INSENSITIVE	INSENSITIVE SCROLL	SCROLL
TYPE_SCROLL_SENSITIVE	SENSITIVE STATIC, SENSITIVE DYNAMIC o ASENSITIVE, dependiendo de la propiedad cursorSensitvity para Connection y DataSource	No soportado

Si un ResultSet de JDBC es estático, el tamaño de la tabla de resultados y el orden de las filas en la tabla de resultados no cambian después de abrir el cursor. Por tanto, si inserta filas en la tabla subyacente, la tabla de resultados para un ResultSet estático no cambia. Si suprime una fila de la tabla de resultados, se produce un hueco por supresión. No puede actualizar ni suprimir un hueco por supresión.

Especificación de la capacidad de actualización, desplazamiento y mantenimiento de ResultSets en aplicaciones JDBC:

Utilice los parámetros especiales de los métodos Connection.prepareStatement o Connection.createStatement para especificar la capacidad de actualización, desplazamiento o retención de ResultSet.

Antes de empezar

Si ha planificado actualizar objetos ResultSet, asegúrese de que la propiedad enableExtendedDescribe no se ha establecido o bien se ha establecido en DB2BaseDataSource.YES (2). Las actualizaciones de objetos ResultSet no funcionan correctamente a menos que se haya habilitado la capacidad de descripción ampliada.

Acerca de esta tarea

Por omisión, los objetos ResultSet son no desplazables y no actualizables. La capacidad de retención por omisión depende de la fuente de datos y se puede determinar a partir del método DatabaseMetaData.getResultSetHoldability. Estos pasos cambian los atributos de capacidad de desplazamiento, actualización y retención para un ResultSet.

Procedimiento

1. Si la sentencia SELECT por la que se define el ResultSet no tiene parámetros de entrada, invoque el método createStatement para crear un objeto Statement. En otro caso, invoque el método prepareStatement para crear un objeto PreparedStatement. Ha de especificar formas de los métodos createStatement o prepareStatement que incluyan los parámetros *resultSetType*, *resultSetConcurrency* o *resultSetHoldability*.

   Esta es la modalidad del método createStatement que da soporte a la capacidad de desplazamiento y a la capacidad de actualización:

   ```
   createStatement(int resultSetType,
   int resultSetConcurrency);
   ```

   Esta es la modalidad del método createStatement que da soporte a la capacidad de desplazamiento, la capacidad de actualización y la capacidad de retención:

   ```
   createStatement(int resultSetType,
   int resultSetConcurrency,
    int resultSetHoldability);
   ```

   Esta es la modalidad del método prepareStatement que da soporte a la capacidad de desplazamiento y a la capacidad de actualización:

   ```
   prepareStatement(String  sql, int resultSetType,
    int resultSetConcurrency);
   ```

   Esta es la modalidad del método prepareStatement que da soporte a la capacidad de desplazamiento, la capacidad de actualización y la capacidad de retención:

   ```
   prepareStatement(String  sql, int resultSetType,
    int resultSetConcurrency, int resultSetHoldability);
   ```

   Importante: En una invocación del método prepareStatement en la que el primer parámetro es una sentencia CALL, no puede especificar la capacidad de desplazamiento, actualización o retención de conjuntos de resultados devueltos desde un procedimiento almacenado. Estas características las determina el código de procedimiento almacenado, cuando declara los cursores para los conjuntos de resultados que se devuelven. Si utiliza el método prepareStatement para preparar una sentencia CALL, y la llamada prepareStatement incluye los parámetros de capacidad de desplazamiento, actualización y retención, IBM Data Server Driver para JDBC y SQLJ no utiliza estos valores de parámetro. Un método prepareStatement con parámetros de capacidad de desplazamiento, actualización y retención se aplica únicamente a la preparación de sentencias de SQL, excepto de la sentencia CALL.

   La tabla siguiente contiene una lista de los valores válidos para *resultSetType* y *resultSetConcurrency*.

Tabla 11. Combinaciones válidas de valores de resultSetType y resultSetConcurrency para conjuntos de resultados

Valor de *resultSetType*	**Valor de** *resultSetConcurrency*
TYPE_FORWARD_ONLY	CONCUR_READ_ONLY
TYPE_FORWARD_ONLY	CONCUR_UPDATABLE
TYPE_SCROLL_INSENSITIVE	CONCUR_READ_ONLY
TYPE_SCROLL_SENSITIVE[1]	CONCUR_READ_ONLY
TYPE_SCROLL_SENSITIVE[1]	CONCUR_UPDATABLE

Tabla 11. Combinaciones válidas de valores de resultSetType y resultSetConcurrency para conjuntos de resultados (continuación)

Valor de *resultSetType*	**Valor de** *resultSetConcurrency*

Nota:

1. Este valor no se aplica a las conexiones con IBM Informix.

resultSetHoldability tiene dos valores posibles: `HOLD_CURSORS_OVER_COMMIT` y `CLOSE_CURSORS_AT_COMMIT`. Cualquiera de estos dos valores se puede especificar con cualquier combinación de *resultSetConcurrency* y *resultSetHoldability*. El valor que defina prevalece sobre la capacidad de retención por omisión de la conexión.

Restricción: si el ResultSet es desplazable y se utiliza para seleccionar columnas de una tabla en un servidor de DB2 Database para Linux, UNIX y Windows, la sentencia SELECT que sirve para definir el ResultSet no puede incluir columnas que tengan los tipos de datos siguientes:
- LONG VARCHAR
- LONG VARGRAPHIC
- BLOB
- CLOB
- XML
- Un tipo diferenciado que esté basado en cualquiera de los tipos de datos anteriores de esta lista
- Un tipo estructurado

2. Si la sentencia SELECT tiene parámetros de entrada, invoque métodos setXXX para pasar valores a los parámetros de entrada.
3. Invoque el método executeQuery para obtener la tabla de resultados de la sentencia SELECT en un objeto ResultSet.
4. Para cada fila a la que desee acceder:
   a. Posicione el cursor utilizando uno de los métodos listados en la tabla siguiente.

      Restricción: Si *resultSetType* es `TYPE_FORWARD_ONLY`, solamente es válido ResultSet.next.

Tabla 12. Métodos para posicionar un cursor desplazable en un conjunto de resultados

Método	**Sitúa el cursor**
first[1]	En la primera fila del ResultSet
last[1]	En la última fila del ResultSet
next[2]	En la fila siguiente del ResultSet
previous[1,3]	En la fila anterior del ResultSet
`absolute(int n)`[1,4]	Si *n*>0, en la fila *n* del ResultSet. Si *n*<0 y *m* es el número de filas del ResultSet, en la fila *m*+*n*+*1* del ResultSet.
`relative(int n)`[1,5,6,]	Si *n*>0, en la fila que está *n* filas después de la fila actual. Si *n*<0, en la fila que está situada *n* filas antes de la fila actual. Si *n*=0, en la fila actual.
afterLast[1]	Después de la última fila del ResultSet
beforeFirst[1]	Antes de la primera fila del ResultSet

Tabla 12. Métodos para posicionar un cursor desplazable en un conjunto de resultados (continuación)

Método	Sitúa el cursor

Notas:

1. Este método no es aplicable a las conexiones con IBM Informix.
2. Si el cursor está situado antes de la primera fila del ResultSet, este método posiciona el cursor en la primera fila.
3. Si el cursor está situado después de la última fila del ResultSet, este método posiciona el cursor en la última fila.
4. Si el valor absoluto de *n* es mayor que el número de filas del conjunto de resultados, este método posiciona el cursor después de la última fila si *n* es positivo, o antes de la primera fila si *n* es negativo.
5. El cursor debe estar en una fila válida del ResultSet para poder utilizar este método. Si el cursor está antes de la primera fila o después de la última fila, el método emite una SQLException.
6. Suponga que *m* es el número de filas del ResultSet y *x* es la fila actual del ResultSet. Si *n*>0 y *x*+*n*>*m*, el controlador posiciona el cursor antes de la primera fila. Si *n*<0 y *x*+*n*<1, el controlador posiciona el cursor antes de la primera fila.

b. Si necesita conocer la posición actual del cursor, utilice el método getRow, isFirst, isLast, `isBeforeFirst` o isAfterLast para obtener esa información.

c. Si para *resultSetType* ha especificado un valor de `TYPE_SCROLL_SENSITIVE` en el paso 1 en la página 63 y necesita ver los valores más recientes de la fila actual, invoque el método refreshRow.

   Recomendación: Debido a que la renovación de las filas de un ResultSet puede afectar negativamente al rendimiento de las aplicaciones, debe invocar refreshRow *solo* cuando necesite ver los datos más recientes.

d. Ejecute una o más de las operaciones siguientes:

   - Para recuperar datos de cada columna de la fila actual del objeto ResultSet, utilice métodos getXXX.
   - Para actualizar la fila actual de la tabla subyacente, utilice métodos updateXXX para asignar valores de columna a la fila actual del ResultSet. Luego utilice updateRow para actualizar la fila correspondiente de la tabla subyacente. Si decide no actualizar la tabla subyacente, invoque el método cancelRowUpdates en lugar del método updateRow.

     El valor *resultSetConcurrency* del ResultSet debe ser `CONCUR_UPDATABLE` para poder utilizar estos métodos.
   - Para suprimir la fila actual de la tabla subyacente, utilice el método deleteRow. La invocación de deleteRow hace que el controlador sustituya la fila actual del ResultSet espacio vacío.

     El valor *resultSetConcurrency* del ResultSet debe ser `CONCUR_UPDATABLE` para utilizar este método.

5. Invoque el método close para cerrar el objeto ResultSet.
6. Invoque el método close para cerrar el objeto Statement o PreparedStatement.

Ejemplo

El código siguiente recupera todas las filas de la tabla de empleados en orden inverso, y actualiza el número de teléfono para el número de empleado "000010". Los números situados a la derecha de determinadas sentencias corresponden a pasos descritos anteriormente.

```
String s;
String stmtsrc;
Connection con;
Statement stmt;
ResultSet rs;
...
stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                           ResultSet.CONCUR_UPDATABLE);                    1
                                       // Crear un objeto Statement
                                       // para un Resultset desplazable
                                       // y actualizable
stmtsrc = "SELECT EMPNO, PHONENO FROM EMPLOYEE " +
          "FOR UPDATE OF PHONENO";
rs = stmt.executeQuery(stmtsrc);       // Crear ResultSet                 3
rs.afterLast();                        // Posicionar el cursor al final
                                       // del conjunto de resultados      4a
while (rs.previous()) {                // Retroceder el cursor
 s = rs.getString("EMPNO");            // Recuperar el número de empleado 4d
                                       // (columna 1 de la tabla de
                                       // resultados)
 System.out.println("Employee number = " + s);
                                       // Imprimir el valor de columna
 if (s.compareTo("000010") == 0) {     // Buscar empleado 000010
  rs.updateString("PHONENO","4657");   // Actualizar su número de teléfono
  rs.updateRow();                      // Actualizar la fila
 }
}
rs.close();                            // Cerrar el conjunto de resultados 5
stmt.close();                          // Cerrar la sentencia              6
```

Figura 15. Uso de un cursor desplazable

Operaciones de SQL sobre varias filas en aplicaciones JDBC:

IBM Data Server Driver para JDBC y SQLJ es compatible con las operaciones INSERT, UPDATE y FETCH sobre varias filas para las conexiones con fuentes de datos que son compatibles con esas operaciones.

INSERT de varias filas

En las aplicaciones JDBC, cuando se ejecutan sentencias INSERT o MERGE que utilizan marcadores de parámetro en un lote, si el servidor de datos soporta la INSERT de varias filas, IBM Data Server Driver para JDBC y SQLJ puede transformar las operaciones INSERT o MERGE por lote en sentencias INSERT de varias filas. Las operaciones INSERT de varias filas pueden ofrecer un mejor rendimiento de las maneras siguientes:

- Para las aplicaciones locales, las inserciones de varias filas producen menos accesos del servidor de datos.
- Para las aplicaciones distribuidas, las inserciones de varias filas producen menos operaciones de red.

No puede ejecutar una operación de inserción de varias filas mediante la inclusión de una sentencia INSERT de varias filas en una serie de sentencia en una aplicación JDBC.

Se utiliza por omisión INSERT de varias filas. Puede utilizar la propiedad enableMultiRowInsertSupport de Connection o DataSource para controlar si se utiliza la INSERT de varias filas. No se puede utilizar INSERT de varias filas para las sentencias INSERT FROM SELECT que se ejecutan en un lote.

FETCH de varias filas

La sentencia FETCH de varias filas puede proporcionar un mejor rendimiento respecto a la recuperación de filas individuales mediante cada sentencia FETCH. Para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS, la sentencia FETCH de varias filas se puede utilizar para cursores de solo avance y cursores desplazables. Para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4, la sentencia FETCH de varias filas se puede utilizar solamente para cursores desplazables.

Cuando el usuario recupera datos en las aplicaciones, IBM Data Server Driver para JDBC y SQLJ determina si se deben utilizar sentencias FETCH de varias filas, dependiendo de varios factores:

- Establecimiento de la propiedad enableRowsetSupport
- Establecimiento de la propiedad useRowsetCursor, para las conexiones con DB2 para z/OS
- El tipo de conectividad de IBM Data Server Driver para JDBC y SQLJ que se está utilizando
- La versión de IBM Data Server Driver para JDBC y SQLJ

Para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 con DB2 para z/OS, uno de los siguientes conjuntos de condiciones debe cumplirse para utilizar la sentencia FETCH de varias filas.

- Primer conjunto de condiciones:
  - La versión de IBM Data Server Driver para JDBC y SQLJ es la 3.51 o posterior.
  - El valor de la propiedad enableRowsetSupport es com.ibm.db2.jcc.DB2BaseDataSource.YES (1) **o** el valor de la propiedad enableRowsetSupport es com.ibm.db2.jcc.DB2BaseDataSource.NOT_SET (0) y el valor de la propiedad useRowsetCursor es `true`.
  - La operación FETCH utiliza un cursor desplazable.

    Para cursores de solo avance, la captación de varias filas puede tener lugar a través de la operación FETCH del bloque DRDA. Sin embargo, este comportamiento no utiliza la posibilidad FETCH de varias filas de la fuente de datos.
- Segundo conjunto de condiciones:
  - La versión de IBM Data Server Driver para JDBC y SQLJ debe ser la 3.1.
  - El valor de la propiedad useRowsetCursor es com.ibm.db2.jcc.DB2BaseDataSource.YES (1).
  - La operación FETCH utiliza un cursor desplazable.

    Para cursores de solo avance, la captación de varias filas puede tener lugar a través de la operación FETCH del bloque DRDA. Sin embargo, este comportamiento no utiliza la posibilidad FETCH de varias filas de la fuente de datos.

Para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 con DB2 para z/OS, deben cumplirse las siguientes condiciones para utilizar la sentencia FETCH de varias filas.

- La versión de IBM Data Server Driver para JDBC y SQLJ es la 3.51 o posterior.
- El valor de la propiedad enableRowsetSupport es com.ibm.db2.jcc.DB2BaseDataSource.YES (1).
- La operación FETCH utiliza un cursor desplazable o un cursor de solo avance.

Para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS, el usuario puede definir la propiedad maxRowsetSize para controlar el tamaño máximo de un conjunto de filas para cada sentencia.

UPDATE o DELETE de posición de varias filas

IBM Data Server Driver para JDBC y SQLJ es compatible con la ejecución de operaciones UPDATE o DELETE de posición que se ajustan al estándar JDBC 1. Esto supone utilizar el método ResultSet.getCursorName para obtener el nombre del cursor para ResultSet, y definir una sentencia UPDATE o DELETE de posición de esta manera:

```
UPDATE tabla SET col1=valor1,...coln=valorN WHERE CURRENT OF nombre_cursor
DELETE
FROM tabla WHERE CURRENT OF nombre_cursor
```

UPDATE o DELETE de varia filas cuando el valor de useRowsetCursor es `true`: si utiliza la técnica de JDBC 1 para actualizar o suprimir datos en un servidor de bases de datos que es compatible con la operación FETCH de varias filas, y esta operación se habilita mediante la propiedad useRowsetCursor, una sentencia UPDATE o DELETE de posición podría actualizar o suprimir inesperadamente varias filas, en lugar de una sola. Para evitar actualizaciones o supresiones inesperadas, puede realizar una de estas acciones:

- Utilice un conjunto de resultados (`ResultSet`) actualizable para recuperar y actualizar una sola fila cada vez, tal como se muestra en el ejemplo anterior.
- Establezca useRowsetCursor en `false`.

UPDATE o DELETE de varias filas cuando el valor de enableRowsetSupport es com.ibm.db2.jcc.DB2BaseDataSource.YES (1): la técnica de JDBC 1 para actualizar o suprimir datos es incompatible con la operación FETCH de varias filas que se habilita mediante la propiedad enableRowsetSupport.

Recomendación: Si sus aplicaciones utilizan la técnica JDBC 1, actualícelas para utilizar los métodos ResultSet.updateRow o ResultSet.deleteRow de JDBC 2.0 para la actividad de actualización o supresión posicionada.

Comprobación de si la fila actual de un ResultSet es un hueco de supresión o un hueco de actualización en una aplicación JDBC:

Si un ResultSet tiene el atributo TYPE_SCROLL_SENSITIVE y el cursor asociado está definido como SENSITIVE STATIC, es necesario comprobar si existen huecos de supresión o de actualización antes de intentar recuperar filas del ResultSet.

Acerca de esta tarea

Después de abrir un ResultSet definido como SENSITIVE STATIC, el conjunto de resultados no cambia de tamaño. Esto significa que las filas suprimidas son sustituidas por marcadores de posición, también denominados *huecos*. Si las filas actualizadas ya no cumplen los criterios para ser incluidas en el ResultSet, esas filas también pasan a ser huecos. Las filas que son huecos no se pueden recuperar.

Para comprobar si la fila actual de un ResultSet es un hueco por supresión o hueco por actualización, siga estos pasos:

Procedimiento

1. Invoque el método DatabaseMetaData.deletesAreDetected o DatabaseMetaData.updatesAreDetected con el argumento TYPE_SCROLL_SENSITIVE para determinar si la fuente de datos crea huecos para un ResultSet definido como TYPE_SCROLL_SENSITIVE
2. Si el resultado devuelto por DatabaseMetaData.deletesAreDetected o DatabaseMetaData.updatesAreDetected es `true`, lo que significa que la fuente de datos puede crear huecos, invoque el método ResultSet.rowDeleted o ResultSet.rowUpdated para determinar si la fila actual es un hueco por supresión o actualización. Si el método devuelve `true`, la fila actual es un hueco.

Ejemplo

El código de programa siguiente determina si la fila actual es un hueco por supresión.

```
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
  ResultSet.CONCUR_UPDATABLE);
                                       // Crear un objeto Statement
                                       // para un Resultset desplazable
                                       // y actualizable
ResultSet rs =
  stmt.executeQuery("SELECT EMPNO FROM EMPLOYEE FOR UPDATE OF PHONENO");
                                       // Crear el conjunto de resultados
DatabaseMetaData dbmd = con.getMetaData();
                                       // Crear el objeto DatabaseMetaData
boolean dbSeesDeletes =
  dbmd.deletesAreDetected(ResultSet.TYPESCROLL_SENSITIVE);
                                       // ¿La base de datos puede detectar huecos
                                       // por supresión?
rs.afterLast();                        // Posicionar el cursor al final
                                       // el conjunto de resultados
while (rs.previous()) {                // Retroceder el cursor
 if (dbSeesDeletes) {                  // Si se puede detectar
                                          huecos por supresión
   if (!(rs.rowDeleted()))             // Si la fila no es un hueco por supresión
   {
     s = rs.getString("EMPNO");            // Obtener el número de empleado
     System.out.println("Employee number = " + s);
                                       // Imprimir el valor de columna
   }
  }
}
rs.close();                       // Cerrar ResultSet
stmt.close();                              // Cerrar Statement
```

Inserción de una fila en un ResultSet de una aplicación JDBC:

Si un ResultSet tiene el valor CONCUR_UPDATABLE para el atributo *resultSetConcurrency*, puede insertar filas en el ResultSet.

Antes de empezar

Asegúrese de que la propiedad enableExtendedDescribe no se ha establecido o bien se ha establecido en DB2BaseDataSource.YES (2). La inserción de una fila en un ResultSet no funciona a menos que se haya habilitado la capacidad de descripción ampliada.

Procedimiento

1. Efectúe los siguientes pasos para cada fila que desee insertar.

a. Invoque el método ResultSet.moveToInsertRow para crear la fila que desea insertar. La fila se crea en un almacenamiento intermedio fuera del ResultSet.

   Si ya existe un almacenamiento intermedio de inserción, se borran todos los valores antiguos del almacenamiento intermedio.

b. Invoque métodos ResultSet.updateXXX para asignar valores a la fila que desea insertar.

   Se debe asignar un valor como mínimo a una columna en el ResultSet. Si no se hace así, se emite una SQLException cuando se inserta la fila en el ResultSet.

   Si no se asigna un valor a una columna en el ResultSet, al actualizar la tabla subyacente, la fuente de datos inserta el valor por omisión de la columna de la tabla asociada.

   Si se asigna un valor nulo a una columna definida como NOT NULL, el controlador JDBC emite una SQLException.

c. Invoque ResultSet.insertRow para insertar la fila en el ResultSet.

   Después de llamar a ResultSet.insertRow, siempre se borran todos los valores del almacenamiento intermedio de inserción, incluso si ResultSet.insertRow falla.

2. Reposición del cursor dentro del ResultSet.

   Para mover el cursor desde la fila de inserción en el ResultSet, llame a cualquiera de los métodos que colocan el cursor en una fila específica, como ResultSet.first, ResultSet.absolute, o ResultSet.relative. Otra posibilidad es llamar a ResultSet.moveToCurrentRow para que mueva el cursor a la fila en el ResultSet que era la fila actual antes de que se realizase la operación de inserción.

   Después de llamar a ResultSet.moveToCurrentRow, se borran todos los valores del almacenamiento intermedio de inserción.

Ejemplo

El código siguiente muestra la inserción de una fila en un ResultSet que consta de todas las filas de la tabla DEPARTMENT de ejemplo. Después de que se haya insertado la fila, el código coloca el cursor donde se encontraba con anterioridad en el ResultSet de la operación de inserción. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
              ResultSet.CONCUR_UPDATABLE);
ResultSet rs = stmt.executeQuery("SELECT * FROM DEPARTMENT");
rs.moveToInsertRow();                                    1a
rs.updateString("DEPT_NO", "M13");                       1b
rs.updateString("DEPTNAME", "TECHNICAL SUPPORT");
rs.updateString("MGRNO", "000010");
rs.updateString("ADMRDEPT", "A00");
rs.insertRow();                                          1c
rs.moveToCurrentRow();                                   2
```

Comprobación de si la fila actual se insertó en un ResultSet en una aplicación JDBC:

Si un ResultSet es dinámico, puede insertar filas en él. Después de insertar filas en un ResultSet, puede ser necesario conocer qué filas se insertaron.

Acerca de esta tarea

Para comprobar si la fila actual se ha insertado en un ResultSet, siga estos pasos:

Procedimiento

1. Invoque los métodos DatabaseMetaData.ownInsertsAreVisible y DatabaseMetaData.othersInsertsAreVisible para determinar si las inserciones pueden ser visibles para el tipo dado de ResultSet.
2. Si las inserciones pueden ser visibles para el ResultSet, invoque el método DatabaseMetaData.insertsAreDetected para determinar si el tipo dado de ResultSet puede detectar inserciones.
3. Si ResultSet puede detectar inserciones, invoque el método ResultSet.rowInserted para determinar si se insertó la fila actual.

Recuperación de filas como datos de byte en aplicaciones JDBC

Puede utilizar el método DB2ResultSet.getDBRowDataAsBytes para recuperar una fila completa de una tabla como bytes sin procesar, y recuperar los datos de columna de las filas devueltas.

Procedimiento

1. Cree un objeto Statement o PreparedStatement.
2. Invoque el método Statement.executeQuery o el método PreparedStatement.executeQuery para obtener un objeto ResultSet.
3. Convierta el objeto ResultSet en un objeto DB2ResultSet.
4. Repita los pasos siguientes hasta que no quede ninguna fila sin recuperar.
   a. Mueva el cursor a la fila siguiente.
   b. Llame al método DB2ResultSet.getDBRowDataAsBytes para recuperar una matriz Object que contenga los datos de fila.
   c. Convierta el primer elemento de la matriz Object en una matriz de bytes.

      Esta matriz de bytes contiene los datos para cada columna en la fila. Consulte la descripción de getDBRowDataAsBytes en el apartado "Interfaz DB2ResultSet" en la página 568 para obtener información acerca del formato de datos.
   d. Convierta el segundo elemento de la matriz Object en una matriz int.

      Cada entero de esta matriz contiene el desplazamiento en la matriz de bytes de datos de fila del principio de los datos para una columna.
   e. Llame al método DB2ResultSet.getDBRowDescriptor para recuperar una matriz int que contenga los datos de fila.

      Esta matriz contiene información descriptiva sobre cada una de las columnas de la fila. Consulte la descripción de getDBRowDescriptor en el apartado "Interfaz DB2ResultSet" en la página 568 para obtener información acerca del formato de datos.
   f. Utilice el valor de desplazamiento para cada columna para localizar los datos de columna y recuperar cada byte de los datos de columna.
   g. Utilice la información que se devuelve desde DB2ResultSet.getDBRowDescriptor para convertir los bytes en un valor del tipo de columna.

Ejemplo

Imaginemos que la tabla MYTABLE se ha definido de la forma siguiente:

```
CREATE TABLE MYTABLE (
 INTCOL1 INTEGER NOT NULL,
 INTCOL2 INTEGER NOT NULL)
```

El siguiente programa recupera filas de datos como bytes sin procesar y recupera los valores de columna de cada fila devuelta. Los números que aparecen a la derecha de las sentencias corresponden a los pasos descritos anteriormente.

```
import java.sql.*;
import com.ibm.db2.jcc.*;

Connection conn;
...
String sql1="select INTCOL, CHARCOL FROM MYTABLE";
int colSqltype;
int colCcsid
int colLen;
int colRep;
Object obj[];
byte data[];
int returnedInfo[];
int numberOfColumns;
int j;
int offsets[];
byte b1;
byte b2;
byte b3
byte b4;
int intVal;

try {
 Statement stmt = conn.createStatement ();                    1
 DB2ResultSet rs = (DB2ResultSet)stmt.executeQuery(sql1);     2,3
 int rowNum=0;
 while(!rs.isLast())
 {
  rs.next();                                                  4a
  rowNum++;
  obj[] = rs.getDBRowDataAsBytes();                           4b
  //*******************************************
  // Recuperar los datos y desplazamientos.
  // Se muestra el código para comprobar el indicador
  // de fila. Dar por supuesto que el indicador de fila
  // indica que los datos son válidos.
  //*******************************************
  data[]=(byte[])obj[0];                                      4c
  offsets[]= (int [])obj[1];                                  4d
  //*******************************************
  // Recuperar los metadatos para cada columna.
  // El primer elemento de la matriz que
  // getDBRowDescriptor devuelve contiene
  // el número de columnas de la fila.
  //*******************************************
  returnedInfo[] = rs.getDBRowDescriptor();                   4e
  numberOfColumns=returnedInfo[0];
  for(j=0;j<numberOfColumns;j++) {
   //*******************************************
   // Obtener los metadatos para una columna.
   //*******************************************
   colSqltype=returnedInfo[(4*j)+1];
   colCcsid=returnedInfo[(4*j)+2];
   colLen=returnedInfo[(4*j)+3];
   colRep=returnedInfo[(4*j)+4];
   //*******************************************
   // Determinar el tipo de la columna. El código
   // no se muestra aquí.
   //*******************************************
```

```
   ...
   //*****************************************
   // Dar por supuesto que los metadatos indican que una
   // columna es INT y Little Endian.
   // El siguiente código recupera los cuatro bytes
   // del valor y los convierte en un entero.
   //*****************************************
   b1 = data[offsets[j]+5];                                    4f
   b2 = data[offsets[j]+4];
   b3 = data[offsets[j]+3];
   b4 = data[offsets[j]+2];
   intVal = ((0xFF & b1) << 24) | ((0xFF & b2) << 16) |        4g
              ((0xFF & b3) << 8) | (0xFF & b4);
   System.out.print("Row "+rowNum+" column "+(j+1)+" "+intVal);
  }
 }
}
```

Llamada a procedimientos almacenados en aplicaciones JDBC

Para llamar a procedimientos almacenados, invoque métodos contenidos en la clase CallableStatement o PreparedStatement.

Acerca de esta tarea

Los pasos básicos para invocar procedimientos almacenados mediante métodos CallableStatement estándar son:

Procedimiento

1. Invoque el método Connection.prepareCall con la sentencia CALL como argumento para crear un objeto CallableStatement.

   Puede representar parámetros con marcadores de parámetro estándar (?), marcadores de parámetro con nombre o parámetros con nombre. Puede combinar marcadores de parámetro estándar y parámetros con nombre en la misma sentencia CALL, pero no puede combinar marcadores de parámetro con nombre y marcadores de parámetro estándar o parámetros con nombre.

   Restricción: Los tipos de parámetros que están permitidos dependen de si la fuente de datos es compatible con la ejecución dinámica de la sentencia CALL. DB2 para z/OS no es compatible con la ejecución dinámica de la sentencia CALL. Para una llamada a un procedimiento almacenado que reside en un servidor de bases de datos DB2 para z/OS, los parámetros pueden ser marcadores de parámetros o literales, pero no expresiones. Aunque todos los parámetros sean literales, no puede utilizar métodos Statement para ejecutar sentencias CALL. Debe utilizar métodos PreparedStatement o métodos CallableStatement. La tabla siguiente lista los tipos de literales permitidos, y sus correspondientes tipos de JDBC.

Tabla 13. Tipos de literales permitidos en los parámetros de las llamadas a procedimientos almacenados DB2 para z/OS

Tipo de parámetro literal	Tipo de JDBC	Ejemplos
Entero	java.sql.Types.INTEGER	-122, 40022, +27
Decimal de coma flotante	java.sql.Types.DOUBLE	23E12, 40022E-4, +2723E+15, 1E+23, 0E0
Decimal de coma fija	java.sql.Types.DECIMAL	-23.12, 40022.4295, 0.0, +2723.23, 10000000000

Tabla 13. Tipos de literales permitidos en los parámetros de las llamadas a procedimientos almacenados DB2 para z/OS (continuación)

Tipo de parámetro literal	Tipo de JDBC	Ejemplos
Carácter	java.sql.Types.VARCHAR	'Grantham Lutz', 'O''Conner', 'ABcde?z?'
Hexadecimal	java.sql.Types.VARBINARY	X'C1C30427', X'00CF18E0'
Serie de caracteres Unicode	java.sql.Types.VARCHAR	UX'0041', UX'0054006500730074'

Importante: En una invocación del método prepareCall, no puede especificar la capacidad de desplazamiento, actualización o retención de conjuntos de resultados devueltos desde un procedimiento almacenado. Estas características las determina el código de procedimiento almacenado, cuando declara los cursores para los conjuntos de resultados que se devuelven. Si especifica cualquiera de las modalidades de prepareCall que incluyen parámetros de capacidad de desplazamiento, actualización y retención, IBM Data Server Driver para JDBC y SQLJ no utiliza estos valores de parámetro. Un método prepareCall con parámetros de capacidad de desplazamiento, actualización y retención se aplica únicamente a la preparación de sentencias de SQL, excepto de la sentencia CALL.

2. Invoque los métodos CallableStatement.setXXX para pasar valores a los parámetros de entrada (parámetros que se definen como IN o INOUT en la sentencia CREATE PROCEDURE).

   Este paso presupone que está utilizando marcadores de parámetro estándar o parámetros con nombre. Alternativamente, si utiliza marcadores de parámetro con nombre, utilice los métodos exclusivos de IBM Data Server Driver para JDBC y SQLJ para pasar valores a los parámetros de entrada.

   Restricción: Si la fuente de datos no es compatible con la ejecución dinámica de la sentencia CALL, debe especificar los tipos de datos para los parámetros de entrada de la sentencia CALL **exactamente** tal como están especificados en la definición del procedimiento almacenado.

   Restricción: La invocación de los métodos CallableStatement.setXXX para pasar valores a los parámetros OUT no está soportada. No es necesario establecer valores para los parámetros OUT de un procedimiento almacenado porque el procedimiento almacenado no utiliza estos valores.

3. Invoque el método CallableStatement.registerOutParameter para registrar parámetros definidos como OUT en la sentencia CREATE PROCEDURE con tipos de datos específicos.

   Este paso presupone que está utilizando marcadores de parámetro estándar o parámetros con nombre. Alternativamente, si utiliza marcadores de parámetro con nombre, utilice los métodos exclusivos de IBM Data Server Driver para JDBC y SQLJ para registrar parámetros OUT con tipos de datos específicos.

   Restricción: Si las fuentes de datos no dan soporte a la ejecución dinámica de la sentencia CALL, debe especificar los tipos de datos para los parámetros OUT, IN o INOUT de la sentencia CALL **exactamente** tal y como están especificados en la definición del procedimiento almacenado.

4. Invoque uno de los métodos siguientes para llamar al procedimiento almacenado:

CallableStatement.executeUpdate
Invoque este método si el procedimiento almacenado no devuelve conjuntos de resultados.

CallableStatement.executeQuery
Invoque este método si el procedimiento almacenado devuelve un solo conjunto de resultados.

Se puede invocar CallableStatement.executeQuery para un procedimiento almacenado que no devuelve ningún conjunto de resultados si se establece la propiedad allowNullResultSetForExecuteQuery en DB2BaseDataSource.YES (1). En ese caso, CallableStatement.executeQuery devuelve un valor nulo. Este comportamiento no cumple el estándar de JDBC.

CallableStatement.execute
Invoque este método si el procedimiento almacenado devuelve varios conjuntos de resultados, o un número desconocido de conjuntos de resultados.

Restricción: Las fuentes de datos IBM Informix no soportan múltiples conjuntos de resultados.

5. Si el procedimiento almacenado devuelve varios conjuntos de resultados, recupere los conjuntos de resultados.

   Restricción: Las fuentes de datos IBM Informix no soportan múltiples conjuntos de resultados.
6. Invoque los métodos CallableStatement.getXXX para recuperar valores a partir de los parámetros de salida (OUT) o entrada y salida (INOUT).
7. Invoque el método CallableStatement.close para cerrar el objeto CallableStatement cuando termine de utilizar ese objeto.

Ejemplo

Ejemplo: el código de programa siguiente muestra la invocación de un procedimiento almacenado que tiene un solo parámetro de entrada, cuatro parámetros de salida y ningún ResultSet devuelto. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
int ifcaret;
int ifcareas;
int xsbytes;
String errbuff;
Connection con;
CallableStatement cstmt;
ResultSet rs;
...
cstmt = con.prepareCall("CALL DSN8.DSN8ED2(?,?,?,?,?)");              1
                                  // Crear un objeto CallableStatement
cstmt.setString (1, "DISPLAY THREAD(*)");                             2
                                  // Establecer parámetro de entrada (mandato DB2)
cstmt.registerOutParameter (2, Types.INTEGER);                        3
                                  // Registrar parámetros de salida
cstmt.registerOutParameter (3, Types.INTEGER);
cstmt.registerOutParameter (4, Types.INTEGER);
cstmt.registerOutParameter (5, Types.VARCHAR);
cstmt.executeUpdate();            // Invocar procedimiento almacenado  4
ifcaret = cstmt.getInt(2); // Obtener valores de parámetros de salida  6
ifcareas = cstmt.getInt(3);
```

```
xsbytes = cstmt.getInt(4);
errbuff = cstmt.getString(5);
cstmt.close();                                                    7
```

Utilización de parámetros con nombre en las sentencias CALL en aplicaciones JDBC

IBM Data Server Driver para JDBC y SQLJ ofrece varios métodos para utilizar parámetros con nombre al llamar a procedimientos almacenados. La sintaxis que utilizan los parámetros con nombre es distinta de la que usan los marcadores de parámetro con nombre.

Acerca de esta tarea

Puede utilizar parámetros con nombre en uno de los lugares siguientes de una aplicación JDBC, o en ambos lugares:

- En la sentencia CALL

  En el caso de los parámetros con nombre, no es necesario especificar los parámetros en la sentencia CALL en el mismo orden en el que aparecen en la definición del procedimiento almacenado. Además, no tendrá que especificar todos los parámetros en la sentencia CALL. Los parámetros sin especificar tomarán los valores por omisión que están especificados en la definición del procedimiento almacenado.
- En los métodos CallableStatement.setXXX, CallableStatement.getXXX y CallableStatement.registerOutParameter

  Puede facilitar la lectura de sus programas si especifica los nombres de parámetro según aparecen en la definición del procedimiento almacenado, en lugar de especificar las posiciones de los parámetros en la definición.

Para utilizar parámetros con nombre con sentencias CALL, realice los pasos siguientes.

Procedimiento

1. Invoque el método Connection.prepareCall con la sentencia CALL como argumento para crear un objeto CallableStatement.

   Para indicar cada parámetro, puede utilizar el marcador de parámetro (?)o la sintaxis siguiente:

   ```
   nombre-parámetro=>?
   ```

   nombre-parámetro identifica un parámetro de la sentencia CREATE PROCEDURE.

   Puede asignar explícitamente el valor por omisión o el valor nulo a un parámetro con nombre especificando la palabra clave DEFAULT o NULL. En el caso de los parámetros para los que se especifica un valor por omisión en la sentencia CREATE PROCEDURE, puede asignar implícitamente los valores por omisión a los parámetros con nombre omitiendo dichos parámetros desde la sentencia CALL. Sólo puede saltar parámetros si todos los parámetros omitidos cuentan con valores por omisión en la definición del procedimiento almacenado.

   No puede mezclar parámetros con nombre y marcadores de parámetro con nombre en la misma sentencia CALL.
2. Invoque los métodos CallableStatement.setXXX para pasar valores a los parámetros de entrada (parámetros que se definen como IN o INOUT en la sentencia CREATE PROCEDURE).

   Puede asignar valores de cualquiera de las formas siguientes:

- Por posición, utilizando CallableStatement.setXXX(parameterIndex,...)
- Por nombre, utilizando CallableStatement.setXXX(parameterName,...)

  Nombre del parámetro es una serie escrita entre comillas dobles cuyo valor coincide con un nombre de parámetro de la sentencia CREATE PROCEDURE.

3. Invoque el método CallableStatement.registerOutParameter para registrar parámetros definidos como OUT en la sentencia CREATE PROCEDURE con tipos de datos específicos.
4. Invoque CallableStatement.executeUpdate, CallableStatement.executeQuery o CallableStatement.execute para ejecutar el procedimiento almacenado.
5. Si el procedimiento almacenado devuelve varios conjuntos de resultados, obtenga esos resultados.

   Puede registrar los parámetros de salida de cualquiera de las formas siguientes:

   - Por posición, utilizando CallableStatement.registerOutParameter(parameterIndex,...)
   - Por nombre, utilizando CallableStatement.registerOutParameter(parameterName,...)

     Nombre del parámetro es una serie escrita entre comillas dobles cuyo valor coincide con un nombre de parámetro de la sentencia CREATE PROCEDURE.

6. Invoque los métodos CallableStatement.getXXX para recuperar valores a partir de los parámetros de salida (OUT) o entrada y salida (INOUT).

   Puede recuperar valores de cualquiera de las formas siguientes:

   - Por posición, utilizando CallableStatement.getXXX(parameterIndex,...)
   - Por nombre, utilizando CallableStatement.getXXX(parameterName,...)

     Nombre del parámetro es una serie escrita entre comillas dobles cuyo valor coincide con un nombre de parámetro de la sentencia CREATE PROCEDURE.

7. Invoque el método CallableStatement.close para cerrar el objeto CallableStatement cuando termine de utilizar ese objeto.

Ejemplo

El código siguiente muestra la invocación de un procedimiento almacenado que tiene la definición siguiente:

```
CREATE PROCEDURE SALS (
  OUT retcode INTEGER,
  IN lowsal DOUBLE,
  IN medsal DOUBLE,
  IN highsal DOUBLE DEFAULT 100000,
  IN department CHAR(3) DEFAULT '---')
SPECIFIC JDBC_SALS
DYNAMIC RESULT SETS 0
DETERMINISTIC
LANGUAGE JAVA
PARAMETER STYLE JAVA
NO DBINFO
FENCED
THREADSAFE
MODIFIES SQL DATA
PROGRAM TYPE SUB
EXTERNAL NAME 'MYJAR:MyClass.sals'
```

Los parámetros de entrada de la sentencia CALL se representan mediante parámetros con nombre. La llamada al tercer parámetro y al cuarto parámetro se

realiza con los valores por omisión para el procedimiento almacenado. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
int hvRetCode;
 // Variable del lenguaje principal para el parámetro de salida
Connection con;
CallableStatement cstmt;
ResultSet rs;
...

cstmt = con.prepareCall(
  "CALL SALS(retcode=>?,lowsal=>?,medsal=>?,highsal=>DEFAULT)");   1
                                  // Preparar la sentencia Call.
                                  // Utilizar implícitamente el valor
                                  // por omisión para el último parámetro
                                  // omitiéndolo.
cstmt.setDouble ("lowsal", 10000);                                  2
cstmt.setDouble ("medsal", 50000);
cstmt.registerOutParameter ("retcode", Types.INTEGER);              3
                                  // Registrar el parámetro de salida
cstmt.executeUpdate();            // Invocar proced. almacenado     4
hvRetCode = cstmt.getInt("retcode");                                6
System.out.println("Return code from SALS call: " + hvRetCode);
cstmt.close();                                                      7
```

Recuperación de datos desde parámetros de salida de cursor en aplicaciones JDBC

Los procedimientos almacenados de DB2 Database para Linux, UNIX y Windows pueden tener parámetros OUT del tipo cursor. Para recuperar datos de dichos parámetros en aplicaciones JDBC, utilice objetos ResultSet.

Acerca de esta tarea

Para recuperar datos de variables de cursor, realice los pasos siguientes.

Procedimiento

1. Defina un objeto ResultSet para cada parámetro OUT que tiene el tipo de datos cursor.
2. Invoque el método Connection.prepareCall con la sentencia CALL como argumento para crear un objeto CallableStatement.
3. Invoque el método CallableStatement.registerOutParameter para registrar los tipos de datos de parámetros definidos como OUT en la sentencia CREATE PROCEDURE.

   El tipo de datos de los parámetros de salida de tipo cursor es com.ibm.db2.jcc.DB2Types.CURSOR.
4. Invoque el procedimiento almacenado.
5. Invoque el método CallableStatement.getObject para recuperar el ResultSet para cada parámetro de cursor OUT.

   Sólo puede llamar a CallableStatement.getObject o CallableStatement.getString en un parámetro de cursor. Si llama a CallableStatement.getString se devuelve un nombre que está asociado con el conjunto de resultados devuelto para el parámetro.

   Si varios parámetros de cursor OUT hacen referencia al mismo cursor en la fuente de datos, se devuelve la misma instancia de ResultSet para todos los parámetros.
6. Recupere filas del objeto ResultSet para cada parámetro de cursor OUT.
7. Cierre ResultSet.

Si el valor de confirmación automática está establecido en `true`, sólo se producirá una operación de confirmación cuando **todos** los conjuntos de resultados que devuelven los parámetros de salida de tipo cursor o el procedimiento almacenado estén cerrados.

Ejemplo

Un tipo de datos cursor y un procedimiento almacenado tienen las definiciones siguientes:

```
CREATE TYPE myRowType AS ROW (name VARCHAR(128))
CREATE TYPE myCursorType AS myRowType CURSOR
CREATE PROCEDURE MYPROC(IN pempNo VARCHAR(6), OUT pcv1 myCursorType)
  RESULT SETS 0
  LANGUAGE SQL
  BEGIN
     DECLARE c1 CURSOR WITH RETURN FOR
       SELECT  empno FROM EMPLOYEE;
     OPEN c1;
     SET pcv1 = CURSOR FOR SELECT name FROM employee WHERE empNo = pempNo;
     OPEN pcv1;
  END
```

El código siguiente llama al procedimiento almacenado MYPROC y utiliza un objeto ResultSet para recuperar datos del cursor pcv1. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
Connection con;
ResultSet rs = null;                  // Parámetro de salida              1
...
CallableStatement cstmt = conn.prepareCall("CALL MYPROC(?, ?)");       2
String hvEmpNo="000500";
cstmt.setString (1, hvEmpNo);
cstmt.registerOutParameter (2, DB2Types.CURSOR);                       3
cstmt.executeUpdate();                // Invocar proced. almacenado     4
String hvEmpName = null;
rs = (java.sql.ResultSet)cstmt.getObject(2);                           5
while (rs.next()) {                   // Recuperar filas conj. result.  6
  hvEmpName=rs.getString(1);
  System.out.println("Employee name for " + hvEmpNo
    + ": " + hvEmpName);
}
rs.close();                           // Cerrar ResultSet               7
```

Invocación de procedimientos almacenados con parámetros ROW en aplicaciones JDBC

Los procedimientos almacenados de DB2 Database para Linux, UNIX y Windows pueden tener parámetros del tipo ROW. Para actualizar y recuperar datos de dichos parámetros en aplicaciones JDBC, utilice objetos Struct.

Acerca de esta tarea

Para actualizar y recuperar datos en parámetros ROW, realice los pasos siguientes.

Procedimiento

1. Defina un objeto Struct para cada parámetro que tiene el tipo de datos ROW.

   Si está utilizando el SDK para Java Versión 6 o posterior, utilice el método createStruct de la interfaz java.sql.Connection. Si está utilizando una versión anterior del SDK para Java, utilice el objeto createStruct de la interfaz com.ibm.db2.jcc.DB2Connection.

2. Invoque el método Connection.prepareCall con la sentencia CALL como argumento para crear un objeto CallableStatement.
3. Invoque métodos CallableStatement.setXXX para asignar valores a los parámetros IN o INOUT en la sentencia CREATE PROCEDURE.

   Utilice el método CallableStatement.setObject para los parámetros ROW.
4. Invoque el método CallableStatement.registerOutParameter para registrar los tipos de datos de parámetros definidos como OUT en la sentencia CREATE PROCEDURE.

   El tipo de datos de los parámetros de salida de tipo ROW es java.sql.Types.STRUCT.
5. Invoque el procedimiento almacenado.
6. Invoque el método CallableStatement.getObject para recuperar el valor de cada parámetro OUT ROW. Convierta los objetos devueltos como valores java.sql.Struct.
7. Recupere datos del objeto Struct para cada parámetro OUT ROW.

Ejemplo

Un tipo ROW y un procedimiento almacenado tienen las definiciones siguientes:

```
CREATE TYPE MYTYPE.PERSON_T AS ROW
 ID INTEGER
 FIRSTNAME VARCHAR(20)
 LASTNAME VARCHAR(20)
 SALARY INTEGER

CREATE PROCEDURE MYSP.PERSON_SP
 (IN PIN MYTYPE.PERSON_T, OUT POUT MYTYPE.PERSON_T)
 LANGUAGE SQL
 BEGIN
 ...
 END
```

El código siguiente llama al procedimiento almacenado MYSP.PERSON_SP y utiliza objetos Struct para asignar datos al parámetro PIN de tipo ROW y para recuperar datos del parámetro POUT de tipo ROW. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
Connection con;
CallableStatement cstmt;
...
personAttributes = new Object[] {
  new Integer(1), "John", "Doe", new Integer(60000)
};
person = con.createStruct("MYTYPE.PERSON_T", personAttributes);          1
cstmt = con.prepareCall("CALL MYSP.PERSON_SP(?,?)");        2
cstmt.setObject(1, person);                                              3
cstmt.registerOutParameter(2, java.sql.Types.STRUCT);                    4
cstmt.execute();                                                         5
person = (java.sql.Struct)cstmt.getObject(2);                            6
Object[] personAttributes = person.getAttributes();                      7
Integer id = (Integer)personAttributes[0];
String firstName = (String)personAttributes[1];
String lastName = (String)personAttributes[2];
Integer salary = (Integer)personAttributes[3];
cstmt.close();
```

Invocación de procedimientos almacenados con parámetros ARRAY de ROW en aplicaciones JDBC

Los procedimientos almacenados de DB2 Database para Linux, UNIX y Windows pueden tener parámetros del tipo ARRAY de ROW. Para actualizar y recuperar datos de dichos parámetros en aplicaciones JDBC, utilice matrices de objetos Struct.

Acerca de esta tarea

Para actualizar y recuperar datos en parámetros ARRAY de ROW, realice los pasos siguientes.

Procedimiento

1. Defina un objeto Struct para cada fila de cada parámetro de entrada que tenga el tipo de datos ARRAY de ROW.

   Si está utilizando el SDK para Java Versión 6 o posterior, utilice el método createStruct de la interfaz jav.sql.Connection. Si está utilizando una versión anterior del SDK para Java, utilice el objeto createStruct de la interfaz com.ibm.db2.jcc.DB2Connection.
2. Defina un objeto Array para cada matriz de filas.

   Si está utilizando el SDK para Java Versión 6 o posterior, utilice el método createArrayOf de la interfaz jav.sql.Connection. Si está utilizando una versión anterior del SDK para Java, utilice el objeto createArrayOf de la interfaz com.ibm.db2.jcc.DB2Connection.
3. Invoque el método Connection.prepareCall con la sentencia CALL como argumento para crear un objeto CallableStatement.
4. Invoque métodos CallableStatement.setXXX para asignar valores a los parámetros IN o INOUT en la sentencia CREATE PROCEDURE.

   Utilice el método CallableStatement.setArray para los parámetros ARRAY de ROW.
5. Invoque el método CallableStatement.registerOutParameter para registrar los tipos de datos de parámetros definidos como OUT en la sentencia CREATE PROCEDURE.

   El tipo de datos de los parámetros de salida de tipo ARRAY de ROW es java.sql.Types.ARRAY.
6. Invoque el procedimiento almacenado.
7. Invoque el método CallableStatement.getArray para recuperar la matriz de cada parámetro OUT de tipo ARRAY de ROW a un objeto java.sql.Array.
8. Invoque el método java.sql.Array.getArray para recuperar el contenido del objeto java.sql.Array. Convierta los objetos recuperados como matrices java.sql.Struct[].
9. Recupere datos de cada elemento de la matriz de objetos Struct para cada parámetro OUT de tipo ARRAY de ROW.

Ejemplo

Un tipo ROW y un procedimiento almacenado tienen las definiciones siguientes:

```
CREATE TYPE MYTYPE.PERSON_T AS ROW
 ID INTEGER
 FIRSTNAME VARCHAR(20)
 LASTNAME VARCHAR(20)
 SALARY INTEGER

CREATE TYPE MYTYPE.PEOPLE_T AS MYTYPE.PERSON_T ARRAY[10]
```

```
CREATE PROCEDURE MYSP.PEOPLE_SP
 (IN PIN MYTYPE.PEOPLE_T, OUT POUT MYTYPE.PEOPLE_T)
 LANGUAGE SQL
 BEGIN
 ...
 END
```

El código siguiente llama al procedimiento almacenado MYSP.PEOPLE_SP y utiliza matrices de objetos Struct para asignar datos al parámetro PIN de tipo ARRAY de ROW y para recuperar datos del parámetro POUT de tipo ARRAY de ROW. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
Connection con;
CallableStatement cstmt;
...

peopleElements = new java.sql.Struct[2];
personAttributes = new Object[] {
  new Integer(1), "John", "Doe", new Integer(60000)
};
peopleElements[0] =
 con.createStruct("MYTYPE.PERSON_T", personAttributes);                  1
personAttributes = new Object[] {
  new Integer(2), "Jane", "Doe", new Integer(65000)
};
peopleElements[1] =
  con.createStruct("MYTYPE.PERSON_T", personAttributes);
Array people = con.createArrayOf("MYTYPE.PERSON_T", peopleElements);     2
cstmt = con.prepareCall("CALL MYSP.PEOPLE_SP(?,?)");                     3
cstmt.setArray(1, people);                                               4
cstmt.registerOutParameter(2, java.sql.Types.ARRAY);                     5
cstmt.execute();                                                         6
java.sql.Array people = cstmt.getArray(2);                               7
java.sql.Struct[] persons =
  (java.sql.Struct[])people.getArray();                                  8
for (int i = 0; i < persons.length; i++) {                               9
  java.sql.Struct person = persons[i];
  Object[] personAttributes = person.getAttributes();
  Integer id = (Integer)personAttributes[0];
  String firstName = (String)personAttributes[1];
  String lastName = (String)personAttributes[2];
  Integer salary = (Integer)personAttributes[3];
}
```

Invocación de procedimientos almacenados con tipos anidados ROW o ARRAY de ROW en aplicaciones JDBC

En los parámetros de procedimiento almacenado de DB2 Database para Linux, UNIX y Windows, los tipos ROW pueden ser anidados. Se pueden recuperar o actualizar los datos de esos parámetros en las aplicaciones JDBC.

Acerca de esta tarea

Para recuperar o actualizar los datos de los parámetros con tipos ROW o ARRAY de ROW anidado, siga estos pasos.

Procedimiento

1. Defina un objeto Struct para cada uno de los tipos ROW con el anidamiento más profundo o un objeto Struct[] para el tipo ARRAY o ROW con el anidamiento más profundo.

Si está utilizando el SDK para Java Versión 6 o posterior, utilice el método createStruct de la interfaz java.sql.Connection. Si está utilizando una versión anterior del SDK para Java, utilice el objeto createStruct de la interfaz com.ibm.db2.jcc.DB2Connection.

2. Repita el paso 1 en la página 82 para el siguiente tipo ROW con anidamiento más profundo. Siga con este proceso hasta que haya definido y llenado un objeto Struct para el tipo ROW más externo.
3. Invoque el método Connection.prepareCall con la sentencia CALL como argumento para crear un objeto CallableStatement.
4. Invoque métodos CallableStatement.setXXX para asignar valores a los parámetros IN o INOUT en la sentencia CREATE PROCEDURE.

   Utilice el método CallableStatement.setObject para los parámetros ROW.
5. Invoque el método CallableStatement.registerOutParameter para registrar los tipos de datos de parámetros definidos como OUT en la sentencia CREATE PROCEDURE.

   El tipo de datos de los parámetros de salida de tipo ROW es java.sql.Types.STRUCT.
6. Invoque el procedimiento almacenado.
7. Invoque el método CallableStatement.getObject para recuperar el valor de cada parámetro OUT ROW. Convierta los objetos devueltos como valores java.sql.Struct.
8. Recupere los datos de objeto Struct de cada parámetro OUT ROW y cada tipo ROW anidado dentro de un parámetro ROW.

Ejemplo

Supongamos que los objetos ARRAY y ROW se definen de la manera siguiente:

MYTYPE.SKILLS es un tipo ARRAY.

```
CREATE TYPE MYTYPE.SKILLS AS VARCHAR(20) ARRAY[10]
```

El tipo ROW MYTYPE.PERSON_T contiene un campo con el tipo ARRAY MYTYPE.SKILLS.

```
CREATE TYPE MYTYPE.PERSON_T AS ROW
 ID INTEGER
 FIRSTNAME VARCHAR(20)
 LASTNAME VARCHAR(20)
 JOBSKILLS MYTYPE.SKILLS
```

El tipo ROW MYTYPE.PEOPLE_T es una matriz de objetos que tienen el tipo ROW MYTYPE.PERSON_T.

```
CREATE TYPE MYTYPE.PEOPLE_T AS MYTYPE.PERSON_T ARRAY[10]
```

El tipo ROW MYTYPE.DEPARTMENT contiene un campo que tiene el tipo ARRAY de ROW MYTYPE.PEOPLE_T.

```
CREATE TYPE MYTYPE.DEPARTMENT AS ROW
 (ID INTEGER,
  DEPTNAME VARCHAR(20),
  DEPTPEOPLE MYTYPE.PEOPLE_T)
```

El procedimiento almacenado MYSP.DEPARTMENT_SP tiene dos parámetros, y cada uno de ellos tiene el tipo MYTYPE.DEPARTMENT.

```
CREATE PROCEDURE MYSP.DEPARTMENT_SP
 (IN PIN MYTYPE.DEPARTMENT, OUT POUT MYTYPE.DEPARTMENT)
   LANGUAGE SQL
   BEGIN
   ...
   END
```

En el código de ejemplo siguiente se muestra cómo utilizar objetos Struct para asignar valores a objetos ROW anidados en un parámetro de entrada de procedimiento almacenado y cómo utilizar objetos Struct para recuperar datos de objetos ROW anidados en un parámetro de salida de procedimiento almacenado. Los números situados a la derecha de determinadas sentencias corresponden a pasos descritos anteriormente.

```
Connection con;
...
java.sql.Struct[] people = new java.sql.Struct[3];
                   // Construir una matriz de objetos Struct para la
                   // matriz MYTYPE.PEOPLE_T de objetos ROW
String[] skills1 = {"java", "C++", "java script"};
                   // Crear y llenar la primera matriz del
                   // objeto MYTYPE.SKILLS ARRAY
Object[] attributes1 =
  new Object [] {new Integer (1), "Alpha", "Doe", skills1};
                   // Crear un objeto con el contenido de la primera
                   // fila de entrada que corresponda a MYTYPE.PERSON_T
people[0] = ((com.ibm.db2.jcc.DB2Connection)con).                          1
               createStruct ("PERSON_T", attributes1);
                   // Llena el primer elemento de la matriz de Struct
                   // para entrarlo en el tipo MYTYPE.PEOPLE_T
String[] skills2 = {"java", "C++", "C"};
                   // Crear y llenar la segunda matriz del
                   // objeto MYTYPE.SKILLS ARRAY
Object[] attributes2 =
  new Object [] {new Integer (2), "Beta", "Doe", skills2};
                   // Crear un objeto con el contenido de la segunda
                   // fila de entrada que corresponda a MYTYPE.PERSON_T
people[1] = ((com.ibm.db2.jcc.DB2Connection)con).                          1
               createStruct ("PERSON_T", attributes2);
                   // Llena el segundo elemento de la matriz de Struct
                   // para entrarlo en el tipo MYTYPE.PEOPLE_T
java.sql.Array peopleArray = ((com.ibm.db2.jcc.DB2Connection)con).
                       createArrayOf("MYTYPE.PEOPLE_T", people);
                   // Crear y llenar un objeto Array para
                   // entrarlo en el objeto DEPTPEOPLE
Object[] deptAttributes =
  new Object [] {new Integer (1), "Jcc", peopleArray };
                   // Crear un objeto con el contenido de una
                   // fila de entrada para el parámetro PIN
java.sql.Struct deptStruct =  ((com.ibm.db2.jcc.DB2Connection)con).        2
               createStruct ("DEPARTMENT", deptAttributes);
                   // Crear y llenar un objeto Struct para
                   // entrar en el parámetro PIN
java.sql.CallableStatement cs = con.prepareCall                            3
  ("CALL MYSP.DEPARTMENT_SP (?, ?)");
cs.setObject(1, deptStruct);                                               4
                   // Asignar el objeto de fila al parámetro de entrada PIN
cs.registerOutParameter(2, java.sql.Types.STRUCT);                         5
                   // Registrar el parámetro de salida como tipo STRUCT
cs.executeUpdate();                                                        6
                   // Invocar el procedimiento almacenado
java.sql.Struct outputStruct = (java.sql.Struct)cs.getObject (2);          7
                   // Recuperar el contenido del parámetro de fila POUT
Object[] structAttributes = outputStruct.getAttributes ();                 8
                   // Recuperar una matriz de objetos que contiene los
                   // atributos de la estructura de salida más externa
```

```
int departmentID = (Integer)structAttributes[0];
String departmentName = (String)structAttributes[1];
System.out.println ("The department ID is: " + departmentID);
System.out.println ("The department Name is: " + departmentName);
java.sql.Struct[] departmentPeople =                                    8
  (java.sql.Struct[])structAttributes[2];
                  // Recuperar el contenido de la estructura anidada
java.sql.Struct personStruct;
System.out.println ("The people in the department are: " );
for (int i = 0; i < departmentPeople.length; i++) {
                  // Recuperar los elementos de la matriz interna
                  // de filas
  personStruct = departmentPeople[i];
  structAttributes = personStruct.getAttributes ();
  System.out.println(
    "id> " + (Integer)structAttributes[0] + " : " +
    "firstName> " + (String)structAttributes[1] + " : " +
    "lastName> " + (String)structAttributes[2] + " : " +
    "skills> ");
  for (int j = 0; j < ((String[]) structAttributes[3]).length; j++)
    System.out.print ("--" + ((String[])structAttributes[3])[j]);
}
```

Recuperación de varios conjuntos de resultados de un procedimiento almacenado en una aplicación JDBC

Si invoca un procedimiento almacenado que devuelve conjuntos de resultados, es necesario que incluya código para recuperar los conjuntos de resultados.

Acerca de esta tarea

Los pasos que debe emprender dependen de si conoce el número de conjuntos de resultados que se devuelven y el contenido de esos resultados.

Recuperación de un número conocido de conjuntos resultados a partir de un procedimiento almacenado de una aplicación JDBC:

La recuperación de un número conocido de conjuntos de resultados a partir de un procedimiento almacenado es una tarea más simple que recuperar un número desconocido de conjuntos resultados.

Acerca de esta tarea

Siga estos pasos para recuperar conjuntos de resultados cuando conoce su número y contenido:

Procedimiento

1. Invoque los métodos Statement.execute, PreparedStatement.execute o CallableStatement.execute para invocar el procedimiento almacenado.

   Utilice PreparedStatement.execute si el procedimiento almacenado tiene parámetros de entrada.
2. Invoque el método getResultSet para obtener el primer conjunto de resultados, que está contenido en un objeto ResultSet.
3. En un bucle, posicione el cursor utilizando el método next y recupere datos de cada columna de la fila actual del objeto ResultSet utilizando métodos getXXX.
4. Si existen *n* conjuntos de resultados, repita los pasos siguientes *n*-1 veces:
   a. Invoque el método getMoreResults para cerrar el conjunto de resultados actual y apuntar al conjunto de resultados siguiente.

b. Invoque el método getResultSet para obtener el conjunto de resultados siguiente, que está contenido en un objeto ResultSet.
c. En un bucle, posicione el cursor utilizando el método next y recupere datos de cada columna de la fila actual del objeto ResultSet utilizando métodos getXXX.

Ejemplo

Ejemplo: el código de programa siguiente muestra la recuperación de dos conjuntos de resultados. El primer conjunto de resultados contiene una columna INTEGER y el segundo conjunto de resultados contiene una columna CHAR. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
CallableStatement cstmt;
ResultSet rs;
int i;
String s;
...
cstmt.execute();                  // Invocar procedimiento almacenado        1
rs = cstmt.getResultSet();        // Obtener primer conjunto de resultados   2
while (rs.next()) {               // Situar el cursor                        3
 i = rs.getInt(1);                // Recuperar el valor del conjunto
                                  // de resultados actual
 System.out.println ("Valor del primer conjunto de resultados = " + i);
                                        // Imprimir el valor
}
cstmt.getMoreResults();           // Apuntar al segundo conj. de resultados  4a
                                        // y cerrar el primer conj. result.
rs = cstmt.getResultSet();        // Obtener segundo conjunto de resultados  4b
while (rs.next()) {               // Situar el cursor                        4c
 s = rs.getString(1);             // Recuperar el valor del conjunto
                                  // de resultados actual
 System.out.println ("Valor del segundo conjunto de resultados = " + s);
                                        // Imprimir el valor
}
rs.close();                       // Cerrar el conjunto de resultados
cstmt.close();                    // Cerrar la sentencia
```

Recuperación de un número desconocido de conjuntos resultados a partir de un procedimiento almacenado de una aplicación JDBC:

La recuperación de un número desconocido de conjuntos de resultados a partir de un procedimiento almacenado es una tarea más compleja que recuperar un número conocido de conjuntos resultados.

Acerca de esta tarea

Para recuperar conjuntos de resultados cuando no conoce su número o su contenido, es necesario ir recuperando losResultSet hasta que no se devuelva ninguno más. Para cada ResultSet, utilice métodos de ResultSetMetaData para determinar su contenido.

Después de invocar un procedimiento almacenado, siga estos pasos básicos para recuperar el contenido de conjuntos de resultados cuando desconoce su número.

Procedimiento

1. Examine el valor devuelto por la sentencia execute mediante la que se invocó el procedimiento almacenado.

Si el valor devuelto es `true`, existe al menos un conjunto de resultados, por lo que necesita ir al paso siguiente.

2. Ejecute en bucle los pasos siguientes:
   a. Invoque el método getResultSet para obtener un conjunto de resultados, que está contenido en un objeto ResultSet. La invocación de este método cierra el conjunto de resultados anterior.
   b. Utilice métodos ResultSetMetaData para determinar el contenido del ResultSet y recuperar datos de él.
   c. Invoque el método getMoreResults para determinar si existe otro conjunto de resultados. Si getMoreResults devuelve `true`, vaya al paso 1 en la página 86 para obtener el conjunto de resultados siguiente.

Ejemplo

Ejemplo: el código de programa siguiente muestra la recuperación de conjuntos de resultados cuando no conoce su número o contenido. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
CallableStatement cstmt;
ResultSet rs;
...
boolean resultsAvailable = cstmt.execute(); // Llamar al procedimiento almacenado
while (resultsAvailable) {                  // Comprobar si hay conjuntos resultados 1
 ResultSet rs = cstmt.getResultSet();       // Obtener conjunto de resultados        2a
 ...                                          // Procesar conjunto de resultados tal
                                            // como procesaría un conjunto de
                                            // resultados de una tabla
 resultsAvailable = cstmt.getMoreResults(); // Buscar conjunto resultados siguiente 2c
                                            // (También cierra el conjunto
                                            // de resultados anterior)
}
```

Mantenimiento de conjuntos de resultados abiertos al recuperar varios conjuntos de resultados a partir de un procedimiento almacenado en una aplicación JDBC:

Existe una modalidad del método getMoreResults que permite dejar abierto el ResultSet actual cuando se abre el siguiente.

Acerca de esta tarea

Para especificar si los conjuntos de resultados deben permanecer abiertos, siga este proceso:

Procedimiento

Cuando invoque getMoreResults para inspeccionar el siguiente ResultSet, utilice esta modalidad del mandato:

```
CallableStatement.getMoreResults(int actual);
```

- Para mantener abierto el ResultSet actual cuando inspecciona el siguiente ResultSet, especifique el valor `Statement.KEEP_CURRENT_RESULT` para *actual*.
- Para cerrar el ResultSet actual cuando inspecciona el siguiente ResultSet, especifique el valor `Statement.CLOSE_CURRENT_RESULT` para *actual*.
- Para cerrar **todos** los objetos ResultSet, especifique el valor `Statement.CLOSE_ALL_RESULTS` para *actual*.

Ejemplo

Ejemplo: el código de programa siguiente mantiene abiertos todos los ResultSet hasta que se ha recuperado el último y luego los cierra.

```
CallableStatement cstmt;
...
boolean resultsAvailable = cstmt.execute(); // Llamar al procedimiento almacenado
if (resultsAvailable==true) {               // Probar conjuntos de resultados
 ResultSet rs1 = cstmt.getResultSet();      // Obtener un conjunto de resultados
 ...
 resultsAvailable = cstmt.getMoreResults(Statement.KEEP_CURRENT_RESULT);
                                            // Buscar siguiente conjunto de
                                            // resultados pero no cerrar conj.
                                            // de resultados anterior
 if (resultsAvailable==true) {              // Buscar otro conjunto de resultados
  ResultSet rs2 = cstmt.getResultSet();     // Obtener siguiente conj. de resultados
  ...                                         // Procesar cualquiera de los dos
                                            // conjuntos de resultados
 }
}
resultsAvailable = cstmt.getMoreResults(Statement.CLOSE_ALL_RESULTS);
                                            // Cerrar los conjuntos de resultados
```

Obtención de información acerca de nombres de parámetro de procedimiento almacenado mediante métodos DB2ParameterMetaData:

El método DB2ParameterMetaData.getProcedureParameterName le permite recuperar el nombre definido de un parámetro en una sentencia CALL de SQL.

Acerca de esta tarea

Para invocar ParameterMetaData.getProcedureParameterName, debe seguir estos pasos básicos:

Procedimiento

1. Invoque el método Connection.prepareCall con la sentencia CALL como argumento para crear un objeto CallableStatement.
2. Pase valores a los parámetros de entrada (parámetros que se definen como IN o INOUT en la sentencia CREATE PROCEDURE).
3. Registre parámetros definidos como OUT en la sentencia CREATE PROCEDURE con tipos de datos específicos.
4. Invoque el procedimiento almacenado.
5. Invoque CallableStatement.getParameterMetaData para recuperar información sobre los parámetros de procedimiento almacenado.
6. Convierta el objeto ParameterMetaData recuperado en un objeto DB2ParameterMetaData.
7. Invoque el método DB2ParameterMetaData.getProcedureParameterName para cada parámetro de sentencia CALL para el que necesita recuperar el nombre de parámetro en la sentencia CREATE PROCEDURE.

Ejemplo

El código siguiente demuestra cómo utilizar el métodoDB2ParameterMetaData.getProcedureParameterName para determinar los nombres que corresponden a los marcadores de parámetro estándar en un procedimiento almacenado que se define de la siguiente forma:

```
CREATE PROCEDURE SP
 (OUT PARM CHAR(10), IN CHAR(10))
```

Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
Connection con;
...
CallableStatement cstmt = con.prepareCall("CALL SP(?, ?)");               1
                                  // Crear un objeto CallableStatement
cstmt.setString (2, "INPUT_VALUE");                                       2
                                  // Establecer parámetro de entrada
cstmt.registerOutParameter (1, java.sql.Types.CHAR);                      3
                                  // Registrar el parámetro de salida
cstmt.execute();                  // Invocar procedimiento almacenado     4
DB2ParameterMetaData md =                                                 5,6
  (DB2ParameterMetaData)cstmt.getParameterMetaData ();
md.getProcedureParameterName(1);  // Devuelve "PARM"                      7
md.getProcedureParameterName(2);  // Devuelve "2"
```

El código siguiente demuestra cómo utilizar el métodoDB2ParameterMetaData.getProcedureParameterName para determinar los nombres que corresponden a los marcadores de parámetro con nombre en un procedimiento almacenado que se define de la siguiente forma:

```
CREATE PROCEDURE SP
 (OUT PARM CHAR(10), IN CHAR(10))
```

Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
Connection con;
...
CallableStatement cstmt = con.prepareCall("CALL SP(:output, :input)");    1
                                  // Crear un objeto CallableStatement
((DB2PreparedStatement)cstmt).setJccStringAtName("input", "INPUT_VALUE"); 2
                                  // Establecer parámetro de entrada
((DB2CallableStatement)cstmt).registerJccOutParameterAtName               3
 ("output", java.sql.Types.CHAR);
                                  // Registrar el parámetro de salida
cstmt.execute();                  // Invocar procedimiento almacenado     4
DB2ParameterMetaData md =                                                 5,6
  (DB2ParameterMetaData)cstmt.getParameterMetaData ();
md.getProcedureParameterName(1);  // Devuelve "PARM"                      7
md.getProcedureParameterName(2);  // Devuelve "2"
```

Objetos LOB en aplicaciones JDBC con el controlador IBM Data Server Driver para JDBC y SQLJ

IBM Data Server Driver para JDBC y SQLJ es compatible con métodos para actualizar y recuperar datos de columnas BLOB, CLOB y DBCLOB de una tabla, y para invocar procedimientos almacenados o funciones definidas por el usuario con parámetros BLOB o CLOB.

Modalidad continua progresiva con IBM Data Server Driver para JDBC y SQLJ

Si la fuente de datos soporta la modalidad continua, también conocida como formato de datos dinámico, IBM Data Server Driver para JDBC y SQLJ puede utilizar la modalidad continua para recuperar datos en columnas LOB o XML.

DB2 para z/OS Versión 9.1 y versiones posteriores es compatible con la modalidad continua progresiva para objetos LOB y XML. DB2 Database para Linux, UNIX y

Windows Versión 9.5 y posteriores IBM Informix Versión 11.50 y posteriores, así como DB2 para i V6R1 y posteriores dan soporte a la corriente continua progresiva para los LOB.

Cuando se utiliza la modalidad continua progresiva, la fuente de datos determina dinámicamente la forma más eficiente de devolver datos LOB o XML, de acuerdo con el tamaño de los objetos LOB o XML.

La modalidad continua progresiva es el comportamiento por omisión en los entornos siguientes:

Versión mínima de IBM Data Server Driver para JDBC y SQLJ	Versión mínima del servidor de datos	Tipos de objetos
3.53	DB2 para i V6R1	LOB, XML
3.50	DB2 Database para Linux, UNIX y Windows Versión 9.5	LOB
3.50	IBM Informix Versión 11.50	LOB
3.2	DB2 para z/OS Versión 9	LOB, XML

El comportamiento de modalidad continua progresiva se define en conexiones nuevas utilizando la propiedad `progressiveStreaming` de IBM Data Server Driver para JDBC y SQLJ.

Para DB2 para z/OS Versión 9.1 y posteriores fuentes de datos o DB2 Database para Linux, UNIX y Windows Versión 9.5 y posteriores fuentes de datos, se puede definir el comportamiento de modalidad continua progresiva para conexiones existentes con el método **`DB2Connection.setDBProgressiveStreaming(DB2BaseDataSource.YES)`**. Si llama a **`DB2Connection.setDBProgressiveStreaming(DB2BaseDataSource.YES)`**, todos los objetos ResultSet que se creen durante la conexión utilizarán el comportamiento de modalidad continua progresiva.

Cuando la modalidad continua progresiva está habilitada, puede controlar el momento en que el controlador JDBC materializa los LOB con la propiedad `streamBufferSize`. Si un objeto LOB o XML es menor que o igual al valor `streamBufferSize`, el objeto se materializa.

Importante: Con la modalidad continua progresiva, al recuperar un valor LOB o XML de un ResultSet en una variable de la aplicación, podrá manipular el contenido de dicha variable de la aplicación hasta que mueva el cursor o lo cierre en ResultSet. Tras esta acción, ya no podrá acceder al contenido de la variable de la aplicación. Si lleva a cabo acciones en el LOB de la variable de la aplicación, recibirá una excepción SQLException. Por ejemplo, suponga que la modalidad continua progresiva está habilitada y ejecuta sentencias del tipo siguiente:

```
...
ResultSet rs = stmt.executeQuery("SELECT CLOBCOL FROM MY_TABLE");
rs.next();                          // Recuperar la primera fila de ResultSet
Clob clobFromRow1  = rs.getClob(1);
                          // Colocar el CLOB de la primera columna de
                          // la primera fila en una variable de aplicación
String substr1Clob = clobFromRow1.getSubString(1,50);
                          // Recuperar los primeros 50 bytes de CLOB
rs.next();                          // Mover el cursor hasta la fila siguiente.
                          // clobFromRow1 ya no se encuentra disponible.
// String substr2Clob = clobFromRow1.getSubString(51,100);
```

```
                          // Esta sentencia daría lugar a una excepción
                                    // SQLException
Clob clobFromRow2  = rs.getClob(1);
                          // Colocar el CLOB de la primera columna de
                          // la segunda fila en una variable de aplicación
rs.close();                         // Cerrar el ResultSet.
                          // clobFromRow2 tampoco se encuentra disponible.
```

Una vez que haya ejecutado rs.next() para colocar el cursor en la segunda fila de ResultSet, el valor CLOB de clobFromRow1 dejará de estar disponible. De forma similar, una vez que haya ejecutado rs.close() para cerrar el ResultSet, los valores de clobFromRow1 y clobFromRow2 dejarán de estar disponibles.

Si inhabilita la modalidad continua progresiva, la forma en que IBM Data Server Driver para JDBC y SQLJ maneja los objetos LOB depende del valor de la propiedad `fullyMaterializeLobData`.

El uso de la modalidad continua progresiva es el método más recomendado para la recuperación de datos LOB o XML.

Localizadores de LOB con IBM Data Server Driver para JDBC y SQLJ

IBM Data Server Driver para JDBC y SQLJ puede utilizar localizadores de LOB para recuperar datos de columnas LOB.

Para hacer que JDBC utilice localizadores de LOB para recuperar datos de columnas LOB, establezca la propiedad `fullyMaterializeLobData` en `false` y establezca la propiedad `progressiveStreaming` en NO (`DB2BaseDataSource.NO` en un programa de aplicación).

El efecto de fullyMaterializeLobData depende de si la fuente de datos es compatible con la modalidad continua progresiva y el valor de la propiedad progressiveStreaming:

- Si la fuente de datos no es compatible con localizadores progresivos:

  Si el valor de fullyMaterializeLobData es `true`, los datos LOB se materializarán completamente dentro del controlador JDBC cuando se capte una fila. Si el valor es `false`, los datos LOB se canalizarán. El controlador utiliza localizadores internamente para recuperar datos LOB en forma de bloques a medida que sea necesario. Es muy recomendable que establezca este valor en `false` cuando recupere datos LOB que contengan grandes volúmenes de datos. El valor por omisión es `true`.
- Si el origen de datos soporta la modalidad continua, también conocida como formato de datos dinámico:

  El controlador JDBC no tiene en cuenta el valor de fullyMaterializeLobData si la propiedad progressiveStreaming está establecida en YES (`DB2BaseDataSource.YES` en un programa de aplicación) o la propiedad no está establecida.

`fullyMaterializeLobData` no tiene ningún efecto sobre los parámetros de procedimientos almacenados.

Como ocurre en cualquier otro lenguaje, un localizador de LOB de una aplicación Java está asociado a una sola fuente de datos. No puede utilizar un mismo localizador de LOB para trasladar datos entre dos fuentes de datos diferentes. Para trasladar datos LOB entre dos fuentes de datos, debe materializar los datos LOB cuando los recupera de una tabla de la primera fuente de datos y luego insertar esos datos en la tabla de la segunda fuente de datos.

Operaciones de LOB con IBM Data Server Driver para JDBC y SQLJ

IBM Data Server Driver para JDBC y SQLJ es compatible con métodos para actualizar y recuperar datos de columnas BLOB, CLOB y DBCLOB de una tabla, y para invocar procedimientos almacenados o funciones definidas por el usuario con parámetros BLOB o CLOB.

Entre otras, puede ejecutar las operaciones siguientes sobre datos LOB cuando utiliza IBM Data Server Driver para JDBC y SQLJ:

- Especificar un BLOB o columna como argumento de los siguientes métodos ResultSet para recuperar datos de una columna BLOB o CLOB:

  Para columnas BLOB:
  - getBinaryStream
  - getBlob
  - getBytes

  Para columnas CLOB:
  - getAsciiStream
  - getCharacterStream
  - getClob
  - getString
- Invocar los métodos ResultSet siguientes para actualizar una columna BLOB o CLOB en un ResultSet actualizable:

  Para columnas BLOB:
  - updateBinaryStream
  - updateBlob

  Para columnas CLOB:
  - updateAsciiStream
  - updateCharacterStream
  - updateClob

  Si especifica -1 para el parámetro *length* en cualquiera de los métodos indicados anteriormente, IBM Data Server Driver para JDBC y SQLJ lee los datos de entrada hasta que se acaban.
- Utilizar los métodos PreparedStatement siguientes para establecer los valores de parámetros que corresponden a columnas BLOB o CLOB:

  Para columnas BLOB:
  - setBytes
  - setBlob
  - setBinaryStream
  - setObject, donde el valor del parámetro *Object* es InputStream.

  Para columnas CLOB:
  - setString
  - setAsciiStream
  - setClob
  - setCharacterStream
  - setObject, donde el valor del parámetro *Object* es Reader.

  Si especifica -1 para *length*, IBM Data Server Driver para JDBC y SQLJ lee los datos de entrada hasta que se acaban.
- Recuperar el valor de un parámetro `CLOB` de JDBC utilizando el método CallableStatement.getString.

Restricción: Con IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2, no puede invocar un procedimiento almacenado que tenga parámetros DBCLOB OUT o INOUT.

Si está utilizando el controlador IBM Data Server Driver para JDBC y SQLJ versión 4.0 o posterior, puede realizar las operaciones adicionales siguientes:

- Utilizar métodos ResultSet.updateXXX o PreparedStatement.setXXX para actualizar un BLOB o CLOB con un valor para *length* de hasta 2GB para un BLOB o CLOB. Por ejemplo, estos métodos están definidos para los BLOB:

  ```
  ResultSet.updateBlob(int columnIndex, InputStream x, long length)
  ResultSet.updateBlob(String columnLabel, InputStream x, long length)
  ResultSet.updateBinaryStream(int columnIndex, InputStream x, long length)
  ResultSet.updateBinaryStream(String columnLabel, InputStream x, long length)
  PreparedStatement.setBlob(int columnIndex, InputStream x, long length)
  PreparedStatement.setBlob(String columnLabel, InputStream x, long length)
  PreparedStatement.setBinaryStream(int columnIndex, InputStream x, long length)
  PreparedStatement.setBinaryStream(String columnLabel, InputStream x, long length)
  ```

- Utilizar métodos ResultSet.updateXXX o PreparedStatement.setXXX sin el parámetro *length* cuando actualiza un BLOB o CLOB, para hacer que IBM Data Server Driver para JDBC y SQLJ lea los datos de entrada hasta que se acaben. Por ejemplo:

  ```
  ResultSet.updateBlob(int columnIndex, InputStream x)
  ResultSet.updateBlob(String columnLabel, InputStream x)
  ResultSet.updateBinaryStream(int columnIndex, InputStream x)
  ResultSet.updateBinaryStream(String columnLabel, InputStream x)
  PreparedStatement.setBlob(int columnIndex, InputStream x)
  PreparedStatement.setBlob(String columnLabel, InputStream x)
  PreparedStatement.setBinaryStream(int columnIndex, InputStream x)
  PreparedStatement.setBinaryStream(String columnLabel, InputStream x)
  ```

- Crear un objeto Blob o Clob que no contenga datos, utilizando el método Connection.createBlob o Connection.createClob.
- Materializar un objeto Blob o Clob en el cliente, cuando se utilicen la modalidad continua progresiva o localizadores, mediante el método Blob.getBinaryStream o `Clob.getCharacterStream`.
- Liberar los recursos retenidos por un objeto Blob o Clob, utilizando el método Blob.free o Clob.free.

Tipos de datos Java para recuperar o actualizar datos de columnas LOB en aplicaciones JDBC

Cuando el controlador JDBC no puede determinar inmediatamente el tipo de datos de un parámetro que se utiliza con una columna LOB, es necesario elegir un tipo de datos para el parámetro que sea compatible con el tipo de datos LOB.

Cuando la propiedad deferPrepares tiene el valor "true", y IBM Data Server Driver para JDBC y SQLJ procesa una llamada PreparedStatement.setXXX, el controlador puede necesitar realizar tareas adicionales de proceso para determinar los tipos de datos. Estas tareas adicionales de proceso pueden afectar al rendimiento del sistema.

Parámetros de entrada para columnas BLOB

Para los parámetros IN de columnas BLOB, o parámetros de INOUT que se utilizan como entrada para columnas BLOB, puede utilizar uno de los métodos siguientes:

- Utilice una variable de entrada java.sql.Blob, que se corresponde exactamente con una columna BLOB:

  ```
  cstmt.setBlob(parmIndex, blobData);
  ```

- Utilice una llamada CallableStatement.setObject que especifique que el tipo de datos de destino es BLOB:

```
byte[] byteData = {(byte)0x1a, (byte)0x2b, (byte)0x3c};
cstmt.setObject(parmInd, byteData, java.sql.Types.BLOB);
```

- Utilice un parámetro de entrada de tipo java.io.ByteArrayInputStream con una llamada CallableStatement.setBinaryStream. Un objeto java.io.ByteArrayInputStream es compatible con un tipo de datos BLOB. Para esta llamada es necesario especificar la longitud exacta de los datos de entrada:

```
java.io.ByteArrayInputStream byteStream =
  new java.io.ByteArrayInputStream(byteData);
int numBytes = byteData.length;
cstmt.setBinaryStream(parmIndex, byteStream, numBytes);
```

Parámetros de salida para columnas BLOB

Para los parámetros OUT de columnas BLOB, o parámetros INOUT que se utilizan como salida para columnas BLOB, puede utilizar uno de los métodos siguientes:

- Utilice la llamada CallableStatement.registerOutParameter para especificar que un parámetro de salida es de tipo BLOB. Luego puede recuperar el valor del parámetro e insertarlo en cualquier variable cuyo tipo de datos sea compatible con un tipo de datos BLOB. Por ejemplo, el código siguiente permite recuperar un valor BLOB e insertarlo en una variable byte[]:

```
cstmt.registerOutParameter(parmIndex, java.sql.Types.BLOB);
cstmt.execute();
byte[] byteData = cstmt.getBytes(parmIndex);
```

Parámetros de entrada para columnas CLOB

Para los parámetros IN de columnas CLOB, o parámetros INOUT que se utilizan como entrada para columnas CLOB, puede utilizar uno de los métodos siguientes:

- Utilice una variable de entrada java.sql.Clob, que se corresponde exactamente con una columna CLOB:

```
cstmt.setClob(parmIndex, clobData);
```

- Utilice una llamada CallableStatement.setObject que especifique que el tipo de datos de destino es CLOB:

```
String charData = "CharacterString";
cstmt.setObject(parmInd, charData, java.sql.Types.CLOB);
```

- Utilice uno de los tipos siguientes de parámetros de entrada:
  - Un parámetro de entrada java.io.StringReader con una llamada cstmt.setCharacterStream:

```
java.io.StringReader reader = new java.io.StringReader(charData);
cstmt.setCharacterStream(parmIndex, reader, charData.length);
```

  - Un parámetro java.io.ByteArrayInputStream con una llamada cstmt.setAsciiStream, para datos ASCII:

```
byte[] charDataBytes = charData.getBytes("US-ASCII");
java.io.ByteArrayInputStream byteStream =
  new java.io.ByteArrayInputStream (charDataBytes);
cstmt.setAsciiStream(parmIndex, byteStream, charDataBytes.length);
```

  Para estas llamadas es necesario especificar la longitud exacta de los datos de entrada.

- Utilice un parámetro de entrada de tipo String con una llamada cstmt.setString:

```
cstmt.setString(parmIndex, charData);
```

  Si la longitud de los datos es mayor que 32KB, y el controlador JDBC no tiene información de DESCRIBE sobre el tipo de datos del parámetro, el controlador JDBC asigna el tipo de datos CLOB a los datos de entrada.

- Utilice un parámetro de entrada de tipo String con una llamada cstmt.setObject, y especifique el tipo de datos de destino como VARCHAR o LONGVARCHAR:

  ```
  cstmt.setObject(parmIndex, charData, java.sql.Types.VARCHAR);
  ```

  Si la longitud de los datos es mayor que 32KB, y el controlador JDBC no tiene información de DESCRIBE sobre el tipo de datos del parámetro, el controlador JDBC asigna el tipo de datos CLOB a los datos de entrada.

Parámetros de salida para columnas CLOB

Para los parámetros OUT de columnas CLOB, o parámetros INOUT que se utilizan como salida para columnas CLOB, puede utilizar uno de los métodos siguientes:

- Utilice la llamada CallableStatement.registerOutParameter para especificar que un parámetro de salida es de tipo CLOB. Luego puede recuperar el valor del parámetro en una variable Clob. Por ejemplo:

  ```
  cstmt.registerOutParameter(parmIndex, java.sql.Types.CLOB);
  cstmt.execute();
  Clob clobData = cstmt.getClob(parmIndex);
  ```

- Utilice la llamada CallableStatement.registerOutParameter para especificar que un parámetro de salida es de tipo VARCHAR o LONGVARCHAR:

  ```
  cstmt.registerOutParameter(parmIndex, java.sql.Types.VARCHAR);
  cstmt.execute();
  String charData = cstmt.getString(parmIndex);
  ```

  Esta técnica se debe utilizar solamente si sabe que la longitud de los datos recuperados es menor o igual que 32KB. En otro caso, los datos se truncan.

Identificadores de fila (ROWID) en JDBC con IBM Data Server Driver para JDBC y SQLJ

DB2 para z/OS y DB2 para i son compatibles con la utilización del tipo de datos ROWID para una columna de una tabla de base de datos. Un ROWID es un valor que identifica una fila de una tabla de una forma exclusiva.

Aunque IBM Informix también es compatible con la utilización de identificadores de fila, el tipo de datos de estos identificadores es INTEGER. Puede seleccionar una columna de identificador de fila de IBM Informix y asignarla a una variable cuyo tipo de datos es un entero de cuatro bytes.

Puede utilizar los siguientes de métodos de ResultSet para recuperar datos de una columna ROWID:

- getRowId (JDBC 4.0 y posterior)
- getBytes
- getObject

Puede utilizar el siguiente método ResultSet para actualizar una columna ROWID de un ResultSet actualizable:

- updateRowId (JDBC 4.0 y posterior)

  updateRowId solamente es válido si el sistema de bases de datos de destino permite la actualización de columnas ROWID.

Si está utilizando JDBC 3.0, para getObject, IBM Data Server Driver para JDBC y SQLJ devuelve una instancia de la clase com.ibm.db2.jcc.DB2RowID específica de IBM Data Server Driver para JDBC y SQLJ.

Si está utilizando JDBC 4.0, para getObject, IBM Data Server Driver para JDBC y SQLJ devuelve una instancia de la clase java.sql.RowId.

Puede utilizar los métodos siguientes de PreparedStatement para definir un valor de un parámetro asociado a una columna ROWID:
- setRowId (JDBC 4.0 y posterior)
- setBytes
- setObject

Si está utilizando JDBC 3.0, para setObject, utilice el tipo `com.ibm.db2.jcc.Types.ROWID` específico de IBM Data Server Driver para JDBC y SQLJ o una instancia de la clase `com.ibm.db2.jcc.DB2RowID` como tipo de destino para el parámetro.

Si está utilizando JDBC 4.0, para setObject, utilice el tipo `java.sql.Types.ROWID` o una instancia de la clase `java.sql.RowId` como tipo de destino para el parámetro.

Puede utilizar los métodos CallableStatement siguientes para recuperar una columna ROWID como parámetro de salida de una llamada de procedimiento almacenado:
- getRowId (JDBC 4.0 y posterior)
- getObject

Para invocar un procedimiento almacenado que está definido con un parámetro de salida ROWID, registre ese parámetro para que sea del tipo `java.sql.Types.ROWID`.

Los valores ROWID son válidos para diferentes períodos de tiempo, dependiendo de la fuente de datos en la que esos valores ROWID están definidos. Utilice el método DatabaseMetaData.getRowIdLifetime para determinar el período de tiempo para el que un valor ROWID es válido. La tabla siguiente muestra los valores que se devuelven para las fuentes de datos.

Tabla 14. Valores de DatabaseMetaData.getRowIdLifetime para las fuentes de datos permitidas

Servidor de bases de datos	**DatabaseMetaData.getRowIdLifetime**
DB2 para z/OS	ROWID_VALID_TRANSACTION
DB2 Database para Linux, UNIX y Windows	ROWID_UNSUPPORTED
DB2 para i	ROWID_VALID_FOREVER
IBM Informix	ROWID_VALID_FOREVER

Ejemplo - Utilización de PreparedStatement.setRowId con el tipo de destino java.sql.RowId: suponga que rwid es un objeto RowId. Para definir el parámetro 1, utilice este formato del método setRowId:

```
ps.setRowId(1, rid);
```

Ejemplo - Utilización de ResultSet.getRowId para recuperar un valor ROWID de una fuente de datos: para recuperar un valor ROWID de la primera columna de un conjunto de resultados y colocarlo en el objeto RowId rwid, utilice este formato del método ResultSet.getRowId:

```
java.sql.RowId rwid = rs.getRowId(1);
```

Ejemplo - Utilización de CallableStatement.registerOutParameter con un tipo de parámetro java.sql.Types.ROWID: para registrar el parámetro 1 de una sentencia CALL como perteneciente al tipo de datos `java.sql.Types.ROWID`, utilice este formato del método registerOutParameter:

```
cs.registerOutParameter(1, java.sql.Types.ROWID)
```

Tipos diferenciados en aplicaciones JDBC

Un tipo diferenciado es un tipo de datos definido por el usuario cuya representación interna es un tipo de datos SQL incorporado. Un tipo diferenciado se crea ejecutando la sentencia CREATE DISTINCT TYPE de SQL.

En un programa JDBC, puede crear un tipo diferenciado utilizando el método executeUpdate para ejecutar la sentencia CREATE DISTINCT TYPE. Puede también utilizar executeUpdate para crear una tabla que incluya una columna de ese tipo. Cuando se recuperan datos de una columna del tipo mencionado, o se actualiza una columna de dicho tipo, se utilizan identificadores Java con tipos de datos que corresponden a los tipos incorporados en que se basan los tipos diferenciados.

El ejemplo siguiente crea un tipo diferenciado que está basado en un tipo INTEGER, crea una tabla con una columna de ese tipo, inserta una fila en la tabla y recupera la fila de la tabla:

```
Connection con;
Statement stmt;
ResultSet rs;
String empNumVar;
int shoeSizeVar;
...
stmt = con.createStatement();         // Crear un objeto Statement
stmt.executeUpdate(
  "CREATE DISTINCT TYPE SHOESIZE AS INTEGER");
                                      // Crear tipo diferenciado
stmt.executeUpdate(
  "CREATE TABLE EMP_SHOE (EMPNO CHAR(6), EMP_SHOE_SIZE SHOESIZE)");
                                      // Crear tabla con tipo diferenciado
stmt.executeUpdate("INSERT INTO EMP_SHOE " +
  "VALUES ('000010', 6)");            // Insertar una fila
rs=stmt.executeQuery("SELECT EMPNO, EMP_SHOE_SIZE FROM EMP_SHOE);
                                      // Crear ResultSet para consulta
while (rs.next()) {
  empNumVar = rs.getString(1);        // Obtener número de empleado
  shoeSizeVar = rs.getInt(2);         // Obtener talla calzado (usar int
                                      // porque el tipo subyacente de
                                      // SHOESIZE es INTEGER)
  System.out.println("Número de empleado = " + empNumVar +
    " Talla de zapato = " + shoeSizeVar);
}
rs.close();                         // Cerrar ResultSet
stmt.close();                         // Cerrar Statement
```

Figura 16. Creación y utilización de un tipo diferenciado

Invocación de procedimientos almacenados con parámetros ARRAY en aplicaciones JDBC

Las aplicaciones JDBC que se ejecutan bajo IBM Data Server Driver para JDBC y SQLJ pueden invocar procedimientos almacenados que tienen parámetros ARRAY.

Los parámetros ARRAY se soportan en procedimientos almacenados en DB2 Database para Linux, UNIX y Windows Versión 9.5 y posteriores.

Puede utilizar objetos java.sql.Array como argumentos para llamar a procedimientos almacenados con parámetros de matriz.

Para los parámetros IN o INOUT, utilice el método DB2Connection.createArrayOf (JDBC 3.0 o anterior) o el método Connection.createArrayOf (JDBC 4.0 o posterior) para crear un objeto java.sql.Array. Utilice el método CallableStatement.setArray o el método CallableStatement.setObject para asignar un objeto java.sql.Array a un parámetro de procedimiento almacenado ARRAY.

Puede registrar un parámetro ARRAY para una llamada de procedimiento almacenado especificando java.sql.Types.ARRAY como tipo de parámetro en una llamada CallableStatement.registerOutParameter.

Existen dos formas de recuperar datos de un parámetro ARRAY de salida:

- Utilice el método CallableStatement.getArray para recuperar los datos y colocarlos en un objeto java.sql.Array, y utilice el método java.sql.Array.getArray para recuperar el contenido del objeto java.sql.Array y colocarlo en una matriz Java.
- Utilice el método CallableStatement.getArray para recuperar los datos y colocarlos en un objeto java.sql.Array. Utilice el método java.sql.Array.getResultSet() para recuperar los datos y colocarlos en un objeto ResultSet. Utilice métodos ResultSet para recuperar elementos de la matriz. Cada fila de ResultSet contiene dos columnas:
  - Un índice en la matriz, que comienza en 1
  - El elemento de matriz

Ejemplo: suponga que los parámetros de entrada y salida IN_PHONE y OUT_PHONE del procedimiento almacenado GET_EMP_DATA son matrices que están definidas de esta manera:

```
CREATE TYPE PHONENUMBERS AS VARCHAR(10) ARRAY[5]
```

Invoque GET_EMP_DATA con los dos parámetros.

```
Connection con;
CallableStatement cstmt;
ResultSet rs;
java.sql.Array inPhoneData;
...
cstmt = con.prepareCall("CALL GET_EMP_DATA(?,?)");
                                  // Crear un objeto CallableStatement
cstmt.setObject (1, inPhoneData); // Establecer parámetro de entrada
cstmt.registerOutParameter (2, java.sql.Types.ARRAY);
                                  // Registrar parámetros de salida
cstmt.executeUpdate();            // Invocar el procedimiento almacenado
Array outPhoneData = cstmt.getArray(2);
                                  // Obtener matriz de parámetro de salida
System.out.println("Valores de parámetros de llamada a GET_EMP_DATA: ");
String [] outPhoneNums = (String [])outPhoneData.getArray();
                          // Recuperar datos de salida del objeto de matriz JDBC
                          // y colocarlos en una matriz Java de tipo carácter
for(int i=0; i<outPhoneNums.length; i++) {
  System.out.print(outPhoneNums[i]);
  System.out.println();
}
```

Puntos de salvaguarda en aplicaciones JDBC

Un punto de salvaguarda de SQL representa el estado que tienen datos y esquemas en un momento determinado dentro de una unidad de trabajo. Puede utilizar sentencias de SQL para establecer un punto de salvaguarda, liberar un punto de salvaguarda, y restaurar datos y esquemas al estado representado por el punto de salvaguarda.

IBM Data Server Driver para JDBC y SQLJ soporta los métodos siguientes para utilizar puntos de salvaguarda:

Connection.setSavepoint() o Connection.setSavepoint(String *nombre*)
Establece un punto de salvaguarda. Estos métodos devuelven un objeto Savepoint, que posteriormente se utiliza en operaciones releaseSavepoint o rollback.

Cuando ejecuta cualquiera de estos dos métodos, DB2 ejecuta la modalidad de la sentencia SAVEPOINT que incluye ON ROLLBACK RETAIN CURSORS.

Connection.releaseSavepoint(Savepoint *punto_salvaguarda*)
Libera el punto de salvaguarda especificado y todos los subsiguientes que haya establecidos.

Connection.rollback(Savepoint *punto_salvaguarda*)
Retrotrae trabajos hasta el punto de salvaguarda especificado.

DatabaseMetaData.supportsSavepoints()
Indica si una fuente de datos soporta puntos de salvaguarda.

Puede indicar si los puntos de salvaguarda son únicos llamando al método DB2Connection.setSavePointUniqueOption. Si llama a este método con un valor de `true`, la aplicación no puede determinar más de un punto de salvaguarda con el mismo nombre dentro de la misma unidad de recuperación. Si llama a este método con un valor `false` (valor por omisión), se pueden crear múltiples puntos de salvaguarda con el mismo nombre dentro de la misma unidad de recuperación, pero la creación de un punto de salvaguarda destruye el punto de salvaguarda con el mismo nombre creado previamente.

El ejemplo siguiente muestra cómo establecer un punto de salvaguarda, retrotraer trabajos hasta un punto de salvaguarda y liberar el punto de salvaguarda.

```
Connection con;
Statement stmt;
ResultSet rs;
String empNumVar;
int shoeSizeVar;
...
con.setAutoCommit(false);        // Desactivar la confirmación automática
stmt = con.createStatement();         // Crear un objeto Statement
...                                        // Ejecutar SQL
con.commit();                             // Confirmar la transacción
stmt.executeUpdate("INSERT INTO EMP_SHOE " +
  "VALUES ('000010', 6)");                // Insertar una fila
((com.ibm.db2.jcc.DB2Connection)con).setSavePointUniqueOption(true);
                                          // Indicar que los puntos de
                                          // salvaguarda son únicos dentro
                                          // de una unidad de recuperación
Savepoint savept = con.setSavepoint("savepoint1");
                                          // Crear un punto de salvaguarda
...
stmt.executeUpdate("INSERT INTO EMP_SHOE " +
  "VALUES ('000020', 10)");               // Insertar otra fila
conn.rollback(savept);                    // Retrotraer el trabajo al punto
                                          // después de la primera inserción
...
con.releaseSavepoint(savept);             // Liberar el punto de salvaguarda
stmt.close();                             // Cerrar la sentencia
conn.commit();                            // Confirmar la transacción
```

Figura 17. Establecimiento, retrotracción y liberación de un punto de salvaguarda en una aplicación JDBC

Recuperación de claves de generación automática en aplicaciones JDBC

Mediante IBM Data Server Driver para JDBC y SQLJ, puede recuperar claves de generación automática (también llamadas claves auto-generadas) a partir de una tabla utilizando métodos de JDBC 3.0.

Una *clave generada automáticamente* es cualquier valor que el servidor de datos genera, en lugar de que lo especifique el usuario. Un tipo de clave generada automáticamente es el contenido de una columna de identidad. Una columna de identidad es una columna de tabla que proporciona un mecanismo para que la fuente de datos genere automáticamente un valor numérico para cada fila. Puede definir una columna de clave en una sentencia CREATE TABLE o ALTER TABLE especificando la cláusula AS IDENTITY cuando defina una columna que tenga un tipo numérico exacto con una escala de 0 (SMALLINT, INTEGER, BIGINT,DECIMAL con una escala de cero, o un tipo diferenciado basado en uno de estos tipos).

Para las conexiones con DB2 para z/OS o DB2 Database para Linux, UNIX y Windows, IBM Data Server Driver para JDBC y SQLJ admite la devolución de claves generadas automáticamente para la sentencia UPDATE buscada, la sentencia INSERT, la sentencia DELETE buscada o la sentencia MERGE. Para las sentencias UPDATE, DELETE o MERGE, cualquier columna puede identificarse como clave generada automáticamente, incluso si el servidor de datos no la ha generado. En este caso, los valores de columna que se devuelven son los valores de columna para las filas que modifica la sentencia UPDATE, DELETE o MERGE.

Restricción: Si la propiedad atomicMultiRowInsert de Connection o DataSource está fijada en `DB2BaseDataSource.YES` (1), no puede preparar una sentencia de SQL para la recuperación de claves generadas automáticamente y utilizar el objeto PreparedStatement para actualizaciones de proceso por lotes. IBM Data Server Driver para JDBC y SQLJ versión 3.50 o posterior emite una excepción de SQL (SQLException) cuando el usuario invoca el método addBatch o executeBatch para un objeto PreparedStatement que está preparado para devolver claves generadas automáticamente.

Recuperación de claves autogeneradas para una sentencia INSERT

Con IBM Data Server Driver para JDBC y SQLJ, se pueden utilizar los métodos de JDBC 3.0 para recuperar las claves que se han generado automáticamente al ejecutar una sentencia INSERT.

Acerca de esta tarea

Para recuperar claves generadas automáticamente que se generan mediante una sentencia INSERT, debe seguir estos pasos:

Procedimiento

1. Utilice uno de los métodos siguientes para indicar que desea obtener claves generadas automáticamente:
   - Si piensa utilizar el método PreparedStatement.executeUpdate para insertar filas, invoque uno de estos formatos del método Connection.prepareStatement para crear un objeto PreparedStatement:

El formato siguiente es válido para una tabla en cualquier fuente de datos que sea compatible con columnas de identidad. *sentencia-sql* debe ser una sentencia INSERT de una sola fila.

Restricción: Para IBM Data Server Driver para JDBC y SQLJ versión 3.57 o posterior, no es válido utilizar el formato siguiente para insertar filas en una vista de un servidor de datos DB2 para z/OS.

```
Connection.prepareStatement(sentencia-sql,
  Statement.RETURN_GENERATED_KEYS);
```

Los formatos siguientes solamente son válidos si la fuente de datos soporta las sentencias SELECT FROM INSERT. *sentencia-sql* puede ser una sentencia INSERT de una sola fila o una sentencia INSERT de varias filas. Cuando utilice el primer formato, especifique los nombres de las columnas para las que desee claves generadas automáticamente. Con el segundo formato, especifique las posiciones de las columnas de la tabla para las que desea claves generadas automáticamente.

```
Connection.prepareStatement(sentencia-sql, String [] nombresColumna);
Connection.prepareStatement(sentencia-sql, int [] Índicescolumna);
```

- Si utiliza el método Statement.executeUpdate para insertar filas, invoque uno de estos formatos del método Statement.executeUpdate:

  El formato siguiente es válido para una tabla en cualquier fuente de datos que sea compatible con columnas de identidad. *sentencia-sql* debe ser una sentencia INSERT de una sola fila.

  Restricción: Para IBM Data Server Driver para JDBC y SQLJ versión 3.57 o posterior, no es válido utilizar el formato siguiente para insertar filas en una vista de un servidor de datos DB2 para z/OS.

  ```
  Statement.executeUpdate(sentencia_sql, Statement.RETURN_GENERATED_KEYS);
  ```

  Los formatos siguientes solamente son válidos si la fuente de datos soporta las sentencias SELECT FROM INSERT. *sentencia-sql* puede ser una sentencia INSERT de una sola fila o una sentencia INSERT de varias filas. Cuando utilice el primer formato, especifique los nombres de las columnas para las que desee claves generadas automáticamente. Con el segundo formato, especifique las posiciones de las columnas de la tabla para las que desea claves generadas automáticamente.

  ```
  Statement.executeUpdate(sentencia-sql, String [] nombresColumna);
  Statement.executeUpdate(sentencia-sql, int [] Índicescolumna);
  ```

2. Invoque el método PreparedStatement.getGeneratedKeys o el método Statement.getGeneratedKeys para recuperar un objeto ResultSet que contiene los valores de claves generadas automáticamente.

   Si incluye el parámetro Statement.RETURN_GENERATED_KEYS, el tipo de datos de las claves generadas automáticamente en ResultSet es DECIMAL, con independencia del tipo de datos de la columna correspondiente.

Ejemplo

El código siguiente crea una tabla con una columna de identidad, inserta una fila en la tabla y recupera el valor de la clave generada automáticamente para la columna de identidad. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
import java.sql.*;
import java.math.*;
import com.ibm.db2.jcc.*;

Connection con;
```

```
Statement stmt;
ResultSet rs;
java.math.BigDecimal iDColVar;
...
stmt = con.createStatement();         // Crear un objeto Statement

stmt.executeUpdate(
  "CREATE TABLE EMP_PHONE (EMPNO CHAR(6), PHONENO CHAR(4), " +
    "IDENTCOL INTEGER GENERATED ALWAYS AS IDENTITY)");
                                      // Crear tabla con columna de identidad
stmt.executeUpdate("INSERT INTO EMP_PHONE (EMPNO, PHONENO) " +          1
  "VALUES ('000010', '5555')",        // Insertar una fila
  Statement.RETURN_GENERATED_KEYS);   // Indicar que desea claves
                                      // generadas automáticamente
rs = stmt.getGeneratedKeys();         // Recupera los valores       2
                                      // generados automáticamente en un ResultSet.
                                      // Se devuelve una sola fila.
                                      // Crear ResultSet para consulta
while (rs.next()) {
  java.math.BigDecimal idColVar = rs.getBigDecimal(1);
                                      // Obtener valor de clave generada
                                      // automáticamente
  System.out.println("valor de clave generada automáticamente = " + idColVar);
}
rs.close();                           // Cerrar ResultSet
stmt.close();                         // Cerrar Statement
```

El código siguiente crea una tabla con una columna de identidad, inserta dos filas en la tabla mediante una sentencia INSERT de varias filas y recupera los valores de clave generados automáticamente para la columna de identidad. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
import java.sql.*;
import java.math.*;
import com.ibm.db2.jcc.*;

Connection con;
Statement stmt;
ResultSet rs;
...
stmt = con.createStatement();

stmt.executeUpdate(
  "CREATE TABLE EMP_PHONE (EMPNO CHAR(6), PHONENO CHAR(4), " +
  "IDENTCOL INTEGER GENERATED ALWAYS AS IDENTITY)");
                                      // Crear tabla con columna de identidad
String[] id_col = {"IDENTCOL"};
int updateCount =                                                       1
  stmt.executeUpdate("INSERT INTO EMP_PHONE (EMPNO, PHONENO)" +
  "VALUES ('000010', '5555'), ('000020', '5556')", id_col);
                                      // Insertar dos filas
                                      // Indicar que se desean claves
                                      // generadas automáticamente
rs = stmt.getGeneratedKeys();         // Recupera los valores       2
                                      // generados automáticamente en un ResultSet.
                                      // Se devuelven dos filas.
                                      // Crear ResultSet para consulta
while (rs.next()) {
 int idColVar = rs.getInt(1);
                                      // Obtener valor de clave generada
                                      // automáticamente
 System.out.println("valor de clave generada automáticamente = " + idColVar);
}
stmt.close();
con.close();
```

Recuperación de claves autogeneradas para una sentencia UPDATE, DELETE o MERGE

Con IBM Data Server Driver para JDBC y SQLJ, se pueden utilizar los métodos de JDBC 3.0 para recuperar las claves que se han generado automáticamente al ejecutar una sentencia UPDATE buscada, una sentencia MERGE o una sentencia DELETE buscada.

Acerca de esta tarea

Para recuperar claves generadas automáticamente que se generan mediante una sentencia UPDATE, DELETE o MERGE, debe seguir estos pasos:

Procedimiento

1. Cree una matriz String que contenga los nombres de las columnas desde las que desea devolver las claves generadas automáticamente.

   La matriz debe ser una matriz de nombres de columna y no de índices de columna.
2. Establezca la modalidad de confirmación automática en `false`.
3. Utilice uno de los métodos siguientes para indicar que desea obtener claves generadas automáticamente:
   - Si piensa utilizar el método PreparedStatement.executeUpdate para actualizar, suprimir o fusionar filas, invoque esta forma del método Connection.prepareStatement para crear un objeto PreparedStatement:

     `Connection.prepareStatement(`*`sentencia-sql`*`, String [] `*`nombresColumna`*`);`
   - Si utiliza el método Statement.executeUpdate para actualizar, suprimir o fusionar filas, invoque esta forma del método Statement.executeUpdate:

     `Statement.executeUpdate(`*`sentencia-sql`*`, String [] `*`nombresColumna`*`);`
4. Invoque el método PreparedStatement.getGeneratedKeys o el método Statement.getGeneratedKeys para recuperar un objeto ResultSet que contiene los valores de claves generadas automáticamente.

Ejemplo

Supongamos que una tabla se defina de la siguiente manera y tenga treinta filas:

```
CREATE TABLE EMP_BONUS
  (EMPNO CHAR(6),
   BONUS DECIMAL(9,2))
```

El código siguiente da nombre a la columna EMPNO como una clave generada automáticamente, actualiza las treinta filas en la tabla EMP_BONUS y recupera los valores de EMPNO para las filas actualizadas. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
import java.sql.*;
...
Connection conn;
...
String[] agkNames = {"EMPNO"};              1
int updateCount = 0;
conn.setAutoCommit(false);                  2
PreparedStatement ps =                      3
 conn.prepareStatement("UPDATE EMP_BONUS SET BONUS = " +
 " BONUS + 300.00",agkNames);
updateCount = ps.executeUpdate();
ResultSet rs = ps.getGeneratedKeys();       4
while (rs.next()) {
 String agkEmpNo = rs.getString(1);
```

```
                // Obtener valor de clave generada automáticamente
 System.out.println("Valor de clave generada automáticamente = " + agkEmpNo);
}
ps.close();
conn.close();
```

Utilización de marcadores de parámetro con nombre en aplicaciones JDBC

Puede utilizar marcadores de parámetro con nombre, en lugar de marcadores de parámetro estándar, en los objetos PreparedStatement y CallableStatement para asignar valores a los marcadores de parámetro de entrada. Puede utilizar marcadores de parámetro con nombre, en lugar de marcadores de parámetro estándar, en los objetos CallableStatement para registrar parámetros OUT que tengan marcadores de parámetro con nombre.

Acerca de esta tarea

Las series de SQL que contienen los siguientes elementos de SQL pueden incluir marcadores de parámetro con nombre:
- CALL
- DELETE
- INSERT
- MERGE
- Bloque de PL/SQL
- SELECT
- SET
- UPDATE

Los marcadores de parámetro con nombre facilitan la lectura de sus aplicaciones JDBC. Si dispone de marcadores de parámetro con nombre en una aplicación, ajuste enableNamedParameterMarkers de la propiedad Connection o DataSource de IBM Data Server Driver para JDBC y SQLJ en `DB2BaseDataSource.YES` (1) para dirigir el controlador para aceptar los marcadores de parámetro con nombre y enviarlos a la fuente de datos como marcadores de parámetro estándar.

Si está conectado con un servidor de datos DB2 Database para Linux, UNIX y Windows Versión 9.7 o posterior, las series de SQL pueden incluir bloques de PL/SQL con parámetros con nombre. Para utilizar parámetros con nombre en bloques de PL/SQL, la variable de registro DB2_COMPATIBILITY_VECTOR debe establecerse en el servidor de datos para permitir la compilación PL/SQL.

Utilización de marcadores de parámetro con nombre con objetos PreparedStatement

Puede utilizar marcadores de parámetro con nombre, en lugar de marcadores de parámetro estándar, en los objetos PreparedStatement para asignar valores a los marcadores de parámetro.

Antes de empezar

Para asegurarse de que las aplicaciones con parámetros con nombre funcionen correctamente, independientemente de la versión y el tipo de servidor de datos, antes de utilizar marcadores de parámetro con nombre en sus aplicaciones, defina la propiedad enableNamedParameterMarkers de Connection o DataSource en `DB2BaseDataSource.YES`.

Acerca de esta tarea

Para utilizar marcadores de parámetro con nombre con objetos PreparedStatement, realice los pasos siguientes.

Procedimiento

1. Ejecute el método Connection.prepareStatement en una serie de sentencia de SQL que contenga marcadores de parámetro con nombre. Los marcadores de parámetro con nombre deben ser conformes a las reglas de denominación de las variables del lenguaje principal de SQL.

   No se pueden mezclar marcadores de parámetro con nombre y marcadores de parámetro estándar en la misma serie de sentencia de SQL.

   Los marcadores de parámetro con nombre son sensibles a las mayúsculas y las minúsculas.
2. Para cada marcador de parámetro con nombre, utilice un método DB2PreparedStatement.setJccXXXAtName para asignar un valor a cada parámetro de entrada con nombre.

   Si utiliza el mismo marcador de parámetro con nombre más de una vez en la misma serie de sentencia de SQL, tendrá que llamar al método setJccXXXAtName para ese marcador de parámetro sólo una vez.

   Recomendación: No utilice el mismo marcador de parámetro con nombre más de una vez en la misma serie de sentencia de SQL si la entrada para ese marcador de parámetro es una secuencia. Podría generar resultados inesperados.

   Restricción: No puede utilizar métodos JDBC PreparedStatement.setXXX estándar con los marcadores de parámetro con nombre. De hacerlo, se generará una excepción.
3. Ejecute PreparedStatement.

Ejemplo

El código siguiente utiliza marcadores de parámetro con nombre para actualizar el número de teléfono '4657' del empleado cuyo número de empleado es '000010'. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
Connection con;
PreparedStatement pstmt;
int numUpd;
...
pstmt = con.prepareStatement(
  "UPDATE EMPLOYEE SET PHONENO=:phonenum WHERE EMPNO=:empnum");
                                  // Crear un objeto PreparedStatement       1
((com.ibm.db2.jcc.DB2PreparedStatement)pstmt).setJccStringAtName
  ("phonenum", "4567");
                                  // Asignar un valor al parámetro phonenum  2
((com.ibm.db2.jcc.DB2PreparedStatement)pstmt).setJccStringAtName
  ("empnum", "000010");
                                  // Asignar un valor al parámetro empnum
numUpd = pstmt.executeUpdate();   // Realizar la actualización               3
pstmt.close();                    // Cerrar el objeto PreparedStatement
```

El código siguiente utiliza marcadores de parámetro con nombre para actualizar valores en un bloque PL/SQL. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
Connection con;
PreparedStatement pstmt;
int numUpd;
...
String sql =
  "BEGIN " +
  "  UPDATE EMPLOYEE SET PHONENO = :phonenum WHERE EMPNO = :empnum; " +
  "END;";
pstmt = con.prepareStatement(sql); // Crear un objeto PreparedStatement      1
((com.ibm.db2.jcc.DB2PreparedStatement)pstmt).setJccStringAtName
  ("phonenum", "4567");
                                   // Asignar un valor al parámetro phonenum  2
((com.ibm.db2.jcc.DB2PreparedStatement)pstmt).setJccStringAtName
  ("empnum", "000010");
                                   // Asignar un valor al parámetro empnum
numUpd = pstmt.executeUpdate();   // Realizar la actualización               3
pstmt.close();                    // Cerrar el objeto PreparedStatement
```

Utilización de marcadores de parámetro con nombre con objetos CallableStatement

Puede utilizar marcadores de parámetro con nombre, en lugar de marcadores de parámetro estándar, en los objetos CallableStatement para asignar valores a los parámetros IN o OUT y para registrar los parámetros OUT.

Antes de empezar

Para asegurarse de que las aplicaciones con parámetros con nombre funcionen correctamente, independientemente de la versión y el tipo de servidor de datos, antes de utilizar marcadores de parámetro con nombre en sus aplicaciones, defina la propiedad enableNamedParameterMarkers de Connection o DataSource en `DB2BaseDataSource.YES`.

Acerca de esta tarea

Para utilizar marcadores de parámetro con nombre con objetos CallableStatement, realice los pasos siguientes.

Procedimiento

1. Ejecute el método Connection.prepareCall en una serie de sentencia de SQL que contenga marcadores de parámetro con nombre.

   Los marcadores de parámetro con nombre deben ser conformes a las reglas de denominación de las variables del lenguaje principal de SQL.

   No se pueden mezclar marcadores de parámetro con nombre y marcadores de parámetro estándar en la misma serie de sentencia de SQL.

   Los marcadores de parámetro con nombre son sensibles a las mayúsculas y las minúsculas.
2. Si no conoce los nombres de los marcadores de parámetro con nombre en la sentencia CALL o la modalidad de los parámetros (IN, OUT o INOUT):
   a. Llame al método CallableStatement.getParameterMetaData para obtener un objeto ParameterMetaData con información sobre los parámetros.
   b. Llame al método ParameterMetaData.getParameterMode para recuperar la modalidad de los parámetros.
   c. Convierta el objeto ParameterMetaData en un objeto DB2ParameterMetaData.
   d. Llame al método DB2ParameterMetaData.getParameterMarkerNames para recuperar los nombres de parámetros.

3. Para cada marcador de parámetro con nombre que representa un parámetro OUT, utilice un método DB2CallableStatement.registerJccOutParameterAtName para registrar el parámetro OUT con un tipo de datos.

   Si utiliza el mismo marcador de parámetro con nombre más de una vez en la misma serie de sentencia de SQL, tendrá que llamar al método registerJccOutParameterAtName para ese marcador de parámetro sólo una vez. Todos los parámetros con el mismo nombre se registran como el mismo tipo de datos.

   Restricción: No puede utilizar métodos JDBC CallableStatement.registerOutParameter estándar con los marcadores de parámetro con nombre. De hacerlo, se generará una excepción.
4. Para cada marcador de parámetro con nombre de un parámetro de entrada, utilice un método DB2CallableStatement.setJccXXXAtName para asignar un valor a cada parámetro de entrada con nombre.

   Los métodos setJccXXXAtName proceden de DB2PreparedStatement.

   Si utiliza el mismo marcador de parámetro con nombre más de una vez en la misma serie de sentencia de SQL, tendrá que llamar al método setJccXXXAtName para ese marcador de parámetro sólo una vez.

   Recomendación: No utilice el mismo marcador de parámetro con nombre más de una vez en la misma serie de sentencia de SQL si la entrada para ese marcador de parámetro es una secuencia. Podría generar resultados inesperados.
5. Ejecute CallableStatement.
6. Llame a los métodos CallableStatement.getXXX o DB2CallableStatement.getJccXXXAtName para recuperar los valores de los parámetros de salida.

Ejemplo

El código siguiente muestra la invocación de un procedimiento almacenado que tiene un parámetro de entrada VARCHAR y un parámetro de salida INTEGER, representados por marcadores de parámetro con nombre. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
...
CallableStatement cstmt =
  con.prepareCall("CALL MYSP(:inparm,:outparm)");
                                    // Crear un objeto CallableStatement  1
((com.ibm.db2.jcc.DB2CallableStatement)cstmt).
   registerJccOutParameterAtName("outparm", java.sql.Types.INTEGER);
                                    // Registrar tipo datos parámetro OUT        3
((com.ibm.db2.jcc.DB2CallableStatement)cstmt).setJccStringAtName("inparm", "4567");
                                  // Asignar un valor al parámetro inparm        4

cstmt.executeUpdate();              // Invocar procedimiento almacenado    5
int outssid = cstmt.getInt(2);    // Obtener el valor del parámetro de salida     6
cstmt.close();
```

El código siguiente ilustra el uso de marcadores de parámetro con nombre en un bloque PL/SQL. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
...
// Leer un bloque PL/SQL, y asignarlo a la variable de serie plsql
CallableStatement cstmt = con.prepareCall(plsql);
                                    // Crear un objeto CallableStatement  1
```

```
DB2ParameterMetaData pm =
  (DB2ParameterMetaData)cs.getParameterMetaData();
                                    // Obtener ParameterMetaData, cambiar 2a,2b
                                    // DB2ParameterMetaData
String[] markers = pm.getParameterMarkerNames();
                                    // Obtener nombres de marcador parám. 2c
// Procesar ParameterMetaData. Presuponga que el primer marcador de parámetro
// es el único parámetro OUT, y que tiene un tipo de datos INTEGER.
...
String parameterName = markers[0];   // Obtener nombre de parámetro OUT
((com.ibm.db2.jcc.DB2CallableStatement)cstmt).
   registerJccOutParameterAtName(parameterName, java.sql.Types.INTEGER);
                                    // Registrar tipo datos parámetro OUT       3
// Asignar valor a parám. de entrada  4
...
cstmt.executeUpdate();              // Invocar procedimiento almacenado    5
int outval = cs.getJccIntAtName(parameterName);
// Obtenga el valor de parámetro de salida mediante
el nombre de parámetro  6
cstmt.close();
```

Suministro de información ampliada sobre el cliente a la fuente de datos mediante métodos específicos de IBM Data Server Driver para JDBC y SQLJ

Un conjunto de métodos específicos de IBM Data Server Driver para JDBC y SQLJ proporciona información adicional sobre el cliente al servidor. Esta información se puede utilizar con fines contables, de gestión de la carga de trabajo o de depuración.

Acerca de esta tarea

Se envía información ampliada sobre el cliente al servidor de bases de datos cuando la aplicación realiza una acción que accede al servidor, tal como ejecutar SQL.

En IBM Data Server Driver para JDBC y SQLJ versión 4.0 o posteriores, los métodos específicos de IBM Data Server Driver para JDBC y SQLJ están en desuso. En su lugar debe utilizar java.sql.Connection.setClientInfo.

La tabla siguiente muestra los métodos específicos de IBM Data Server Driver para JDBC y SQLJ.

Tabla 15. Métodos que proporcionan información sobre el cliente al servidor DB2

Método	Información proporcionada
setDB2ClientAccountingInformation	Información contable
setDB2ClientApplicationInformation	Nombre de la aplicación que está trabajando con una conexión
setDB2ClientDebugInfo	El atributo de conexión CLIENT DEBUGINFO para el depurador unificado
setDB2ClientProgramId	Serie de caracteres especificada por el emisor que le permite identificar qué programa está asociado a una determinada sentencia de SQL. setDB2ClientProgramId no se aplica a servidores de datos DB2 Database para Linux, UNIX y Windows.
setDB2ClientUser	Nombre de usuario para una conexión

Tabla 15. Métodos que proporcionan información sobre el cliente al servidor DB2 (continuación)

Método	Información proporcionada
setDB2ClientWorkstation	Nombre de estación de trabajo del cliente para una conexión

Para definir la información ampliada sobre el cliente, siga estos pasos:

Procedimiento

1. Cree un objeto Connection.
2. Convierta el objeto java.sql.Connection en un objeto com.ibm.db2.jcc.DB2Connection.
3. Invoque cualquiera de los métodos mostrados en la Tabla 15 en la página 108.
4. Ejecute una sentencia de SQL para hacer que la información se envíe al servidor DB2.

Ejemplo

El código de programa siguiente ejecuta los pasos anteriores para pasar un nombre de usuario y un nombre de estación de trabajo al servidor DB2. Los números situados a la derecha de determinadas sentencias corresponden a pasos descritos anteriormente.

```
public class ClientInfoTest {
  public static void main(String[] args) {
    String url = "jdbc:db2://sysmvs1.stl.ibm.com:5021/san_jose";
    try {
      Class.forName("com.ibm.db2.jcc.DB2Driver");
      String user = "db2adm";
      String password = "db2adm";
      Connection conn = DriverManager.getConnection(url,          1
        user, password);
      if (conn instanceof DB2Connection) {
        DB2Connection db2conn = (DB2Connection) conn;             2
        db2conn.setDB2ClientUser("Michael L Thompson");           3
        db2conn.setDB2ClientWorkstation("sjwkstn1");
        // Ejecutar SQL para forzar el envío de información
        // ampliada sobre el cliente al servidor
        conn.prepareStatement("SELECT * FROM SYSIBM.SYSDUMMY1"
          + "WHERE 0 = 1").executeQuery();                        4
      }
    } catch (Throwable e) {
        e.printStackTrace();
      }
  }
}
```

Figura 18. Ejemplo de suministro de información ampliada sobre el cliente a un servidor DB2

Suministro de información ampliada sobre el cliente a la fuente de datos mediante propiedades de información del cliente

IBM Data Server Driver para JDBC y SQLJ versión 4.0 es compatible con las propiedades de información del cliente de JDBC 4.0, que puede utilizar para proporcionar información adicional sobre el cliente al servidor. Esta información se puede utilizar con fines contables, de gestión de la carga de trabajo o de depuración.

Acerca de esta tarea

Se envía información ampliada sobre el cliente al servidor de bases de datos cuando la aplicación realiza una acción que accede al servidor, tal como ejecutar SQL.

La aplicación puede también utilizar el método Connection.getClientInfo para recuperar información del cliente a partir del servidor de bases de datos, o ejecutar el método DatabaseMetaData.getClientInfoProperties para determinar qué información de cliente es compatible con el controlador.

Se deben utilizar las propiedades de información del cliente de JDBC 4.0 en lugar de los métodos específicos de IBM Data Server Driver para JDBC y SQLJ, que están en desuso.

Para establecer las propiedades de información del cliente, siga estos pasos:

Procedimiento

1. Cree un objeto Connection.
2. Invoque el método java.sql.Connection.setClientInfo para establecer cualquiera de las propiedades de información del cliente que sean compatibles con el servidor de bases de datos.
3. Ejecute una sentencia de SQL para hacer que la información se envíe al servidor de bases de datos.

Ejemplo

El código de programa siguiente ejecuta los pasos anteriores para pasar un nombre de usuario de cliente y un nombre de sistema principal al servidor DB2. Los números situados a la derecha de determinadas sentencias corresponden a pasos descritos anteriormente.

```
public class ClientInfoTest {
  public static void main(String[] args) {
   String url = "jdbc:db2://sysmvs1.stl.ibm.com:5021/san_jose";
    try {
      Class.forName("com.ibm.db2.jcc.DB2Driver");
      String user = "db2adm";
      String password = "db2adm";
      Connection conn = DriverManager.getConnection(url,       1
        user, password);
      conn.setClientInfo("ClientUser", "Michael L Thompson"); 2
      conn.setClientInfo("ClientHostname, "sjwkstn1");
      // Ejecutar SQL para forzar el envío de información
        // ampliada sobre el cliente al servidor
      conn.prepareStatement("SELECT * FROM SYSIBM.SYSDUMMY1"
         + "WHERE 0 = 1").executeQuery();                     3
    } catch (Throwable e) {
      e.printStackTrace();
    }
  }
}
```

Figura 19. Ejemplo de suministro de información ampliada sobre el cliente a un servidor DB2

Compatibilidad de IBM Data Server Driver para JDBC y SQLJ con las propiedades de información del cliente

JDBC 4.0 incluye propiedades de información del cliente, que contienen información sobre una conexión con una fuente de datos. El método

DatabaseMetaData.getClientInfoProperties devuelve una lista de propiedades de información del cliente que se pueden utilizar con IBM Data Server Driver para JDBC y SQLJ.

Cuando invoca DatabaseMetaData.getClientInfoProperties, se devuelve un conjunto de resultados que contiene las columnas siguientes:
- NAME
- MAX_LEN
- DEFAULT_VALUE
- DESCRIPTION

La tabla siguiente lista los valores de propiedades de información del cliente que IBM Data Server Driver para JDBC y SQLJ devuelve para DB2 Database para Linux, UNIX y Windows y para DB2 para i.

Tabla 16. Valores de propiedades de información del cliente para DB2 Database para Linux, UNIX y Windows y para DB2 para i

NAME	MAX_LEN (bytes)	DEFAULT_VALUE	DESCRIPTION
ApplicationName	255	Serie de caracteres vacía	Nombre de la aplicación que está actualmente haciendo uso de la conexión. Este valor se almacena en el registro especial de DB2 CURRENT CLIENT_APPLNAME.
ClientAccountingInformation	255	Serie de caracteres vacía	Valor de la serie de caracteres contable de la información del cliente que está especificada para la conexión. Este valor se almacena en el registro especial de DB2 CURRENT CLIENT_ACCTNG.
ClientHostname	255	Valor establecido por DB2Connection.setDB2ClientWorkstation. Si el valor no está establecido, el valor por omisión es el nombre del sistema principal local.	Nombre de sistema principal del sistema donde se ejecuta la aplicación que está haciendo uso de la conexión. Este valor se almacena en el registro especial de DB2 CURRENT CLIENT_WRKSTNNAME.
ClientUser	255	Serie de caracteres vacía	Nombre del usuario para el que se ejecuta la aplicación que está haciendo uso de la conexión. Este valor se almacena en el registro especial de DB2 CURRENT CLIENT_USERID.

La tabla siguiente lista los valores de propiedades de información del cliente que IBM Data Server Driver para JDBC y SQLJ devuelve para DB2 para z/OS cuando la conexión utiliza conectividad de tipo 4.

Tabla 17. Valores de propiedades de información de conectividad de tipo 4 para DB2 para z/OS

NAME	MAX_LEN (bytes)	DEFAULT_VALUE	DESCRIPTION
ApplicationName	32	Valor de la propiedad clientProgramName, si está definida. "db2jcc_application" en otro caso.	Nombre de la aplicación que está actualmente haciendo uso de la conexión. Este valor se almacena en el registro especial de DB2 CURRENT CLIENT_APPLNAME.

Tabla 17. Valores de propiedades de información de conectividad de tipo 4 para DB2 para z/OS (continuación)

NAME	MAX_LEN (bytes)	DEFAULT_VALUE	DESCRIPTION
ClientAccountingInformation	200	Serie de caracteres vacía	Valor de la serie de caracteres contable de la información del cliente que está especificada para la conexión. Este valor se almacena en el registro especial de DB2 CURRENT CLIENT_ACCTNG.
ClientHostname	18	Valor establecido por DB2Connection.setDB2ClientWorkstation. Si el valor no está establecido, el valor por omisión es el nombre del sistema principal local.	Nombre de sistema principal del sistema donde se ejecuta la aplicación que está haciendo uso de la conexión. Este valor se almacena en el registro especial de DB2 CURRENT CLIENT_WRKSTNNAME.
ClientUser	16	Valor establecido por DB2Connection.setDB2ClientUser. Si el valor no está establecido, el valor por omisión es el ID de usuario actual utilizado para conectar con la base de datos.	Nombre del usuario para el que se ejecuta la aplicación que está haciendo uso de la conexión. Este valor se almacena en el registro especial de DB2 CURRENT CLIENT_USERID.

La tabla siguiente lista los valores de propiedades de información del cliente que IBM Data Server Driver para JDBC y SQLJ devuelve para DB2 para z/OS cuando la conexión utiliza conectividad de tipo 2.

Tabla 18. Valores de propiedades de información de cliente para conectividad de tipo 2 en DB2 para z/OS

NAME	MAX_LEN (bytes)	DEFAULT_VALUE	DESCRIPTION
ApplicationName	32	La serie "db2jcc_application"	Nombre de la aplicación que está actualmente haciendo uso de la conexión. Este valor se almacena en el registro especial de DB2 CURRENT CLIENT_APPLNAME.
ClientAccountingInformation	200	Serie de caracteres vacía	Valor de la serie de caracteres contable de la información del cliente que está especificada para la conexión. Este valor se almacena en el registro especial de DB2 CURRENT CLIENT_ACCTNG.
ClientHostname	18	La serie "RRSAF"	Nombre de sistema principal del sistema donde se ejecuta la aplicación que está haciendo uso de la conexión. Este valor se almacena en el registro especial de DB2 CURRENT CLIENT_WRKSTNNAME.
ClientUser	16	El ID de usuario que se especificó para la conexión. Si no se ha especificado ningún ID de usuario, se utiliza el ID de usuario de RACF.	Nombre del usuario para el que se ejecuta la aplicación que está haciendo uso de la conexión. Este valor se almacena en el registro especial de DB2 CURRENT CLIENT_USERID.

La tabla siguiente lista los valores de propiedades de información del cliente que IBM Data Server Driver para JDBC y SQLJ devuelve para IBM Informix

Tabla 19. Valores de propiedades de información del cliente para IBM Informix

NAME	MAX_LEN (bytes)	DEFAULT_VALUE	DESCRIPTION
ApplicationName	20	Serie de caracteres vacía	Nombre de la aplicación que está actualmente haciendo uso de la conexión.
ClientAccountingInformation	199	Serie de caracteres vacía	Valor de la serie de caracteres contable de la información del cliente que está especificada para la conexión.
ClientHostname	20	Valor establecido por DB2Connection.setDB2ClientWorkstation. Si el valor no está establecido, el valor por omisión es el nombre del sistema principal local.	Nombre de sistema principal del sistema donde se ejecuta la aplicación que está haciendo uso de la conexión.
ClientUser	1024	Serie de caracteres vacía	Nombre del usuario para el que se ejecuta la aplicación que está haciendo uso de la conexión.

Información sobre parámetros ampliados con IBM Data Server Driver para JDBC y SQLJ

Los métodos y constantes específicos de IBM Data Server Driver para JDBC y SQLJ le permiten asignar el valor por omisión o ningún valor a las columnas de tabla o las columnas ResultSet.

El servidor de datos debe ofrecer soporte para los indicadores ampliados antes de poder utilizar los métodos que proporcionan información sobre indicadores ampliados en las aplicaciones Java. Si invoca uno de estos métodos para un servidor de datos que no ofrece soporte para los indicadores ampliados, se generará una excepción. La información sobre parámetros ampliados recibe soporte en DB2 para z/OS Versión 10 o posterior, o DB2 Database para Linux, UNIX y Windows Versión 9.7 o posterior.

En la tabla siguiente se enumeran los métodos que proporcionan información sobre parámetros ampliados.

Métodos de información sobre parámetros ampliados	Finalidad
DB2PreparedStatement.setDBDefault, DB2PreparedStatement.setJccDBDefaultAtName	Establece un parámetro de entrada en el valor por omisión.
DB2PreparedStatement.setDBUnassigned, DB2PreparedStatement.setJccDBUnassignedAtName	Indica que un parámetro de entrada es UNASSIGNED. Esta acción produce el mismo comportamiento que tendría lugar si el parámetro de entrada no apareciese en el texto de la sentencia de SQL.
DB2ResultSet.updateDBDefault	Establece un valor de columna de la fila ResultSet actual en el valor por omisión.

Estos métodos son aplicables solamente para los marcadores de parámetro que aparecen en uno de los lugares siguientes:

- La lista SET de una sentencia UPDATE
- La lista SET de una sentencia MERGE
- La lista VALUES de una sentencia INSERT
- La lista VALUES de una sentencia MERGE

- La tabla fuente de una sentencia MERGE
- La lista SELECT de una operación INSERT de una sentencia SELECT

Se generará una excepción SQLException si se utilizan estos métodos en cualquier otro contexto.

Como alternativa, se pueden utilizar los métodos estándar PreparedStatement.setObject o ResultSet.updateObject con las constantes específicas DB2PreparedStatement.DB_PARAMETER_DEFAULT o DB2PreparedStatement.DB_PARAMETER_UNASSIGNED de IBM Data Server Driver para JDBC y SQLJ para asignar a los parámetros el valor por omisión o ningún valor.

La información sobre parámetros ampliados puede simplificar los programas de aplicación que tienen diversas variables de entrada, cada una de las cuales puede enviar un valor o el valor por omisión al servidor de datos o no es necesario que aparezca en la sentencia de SQL. En lugar de preparar series de sentencia independientes para todas las combinaciones de valores de variables, puede preparar una sola serie de sentencia. El objeto PreparedStatement resultante puede utilizarse en un lote homogéneo, pero varios objetos PreparedStatement distintos no pueden utilizarse en un lote homogéneo.

Uso de métodos o constantes DB2PreparedStatement para proporcionar información sobre parámetros ampliados

Utilice métodos DB2PreparedStatement o PreparedStatement con constantes DB2PreparedStatement para asignar valores por omisión a columnas de destino o para no asignarles ningún valor.

Acerca de esta tarea

Siga estos pasos para enviar al servidor de datos información ampliada sobre el cliente para una PreparedStatement.

Procedimiento

1. Cree un objeto PreparedStatement.

   La sentencia de SQL es una sentencia INSERT, UPDATE o MERGE.
2. Si no utiliza setObject para asignar los valores, convierta el objeto PreparedStatement a un objeto com.ibm.db2.jcc.DB2PreparedStatement.
3. Invoque uno de los métodos siguientes:
   - Si no utiliza setObject para asignar el valor:
     - Para asignar el valor por omisión de la columna de destino al parámetro de entrada, invoque DB2PreparedStatement.setDBDefault o DB2PreparedStatement.setJccDBDefaultAtName.
     - Para marcar el parámetro de entrada como UNASSIGNED, invoque DB2PreparedStatement.setDBUnassigned o DB2PreparedStatement.setJccDBUnassignedAtName.
   - Si utiliza setObject para asignar el valor:
     - Para asignar el valor por omisión de la columna de destino al parámetro de entrada, invoque PreparedStatement.setObject con DB2PreparedStatement.DB_PARAMETER_DEFAULT como valor asignado.

- Para marcar el parámetro de entrada como UNASSIGNED, invoque PreparedStatement.setObject con DB2PreparedStatement.DB_PARAMETER_UNASSIGNED como valor asignado.

4. Ejecute la sentencia de SQL.

Ejemplo

El código siguiente asigna los valores por omisión de las columnas de destino del tercer y quinto parámetro en una sentencia INSERT. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
import java.sql.*;
import com.ibm.db2.jcc.*;

Connection conn;
...
PreparedStatement p = conn.prepareStatement(                    1
  "INSERT INTO DEPARTMENT " +
  "(DEPTNO, DEPTNAME, MGRNO, ADMRDEPT, LOCATION) " +
  "VALUES (?,?,?,?,?)");
p.setString(1, "X00");
p.setString(2, "FACILITIES");
p.setString(4, "A00");
((com.ibm.db2.jcc.DB2PreparedStatement)p).setDBDefault(3);      2,3
((com.ibm.db2.jcc.DB2PreparedStatement)p).setDBDefault(5);
int uCount = p.executeUpdate();                                 4
...
p.close();                          // Close PreparedStatement
```

El código siguiente utiliza el método PreparedStatement.setObject y las constantes DB2PreparedStatement para llevar a cabo la misma función. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
import java.sql.*;
import com.ibm.db2.jcc.*;

Connection conn;
...

PreparedStatement p = conn.prepareStatement(                    1
  "INSERT INTO DEPARTMENT " +
  "(DEPTNO, DEPTNAME, MGRNO, ADMRDEPT, LOCATION) " +
  "VALUES (?,?,?,?,?)");
p.setString(1, "X00");
p.setString(2, "FACILITIES");
p.setString(4, "A00");
p.setObject(3, DB2PreparedStatement.DB_PARAMETER_DEFAULT);      3
p.setObject(5, DB2PreparedStatement.DB_PARAMETER_DEFAULT);
int uCount = p.executeUpdate();                                 4
...
p.close();                          // Close PreparedStatement
```

En estos ejemplos, el uso del método DB2PreparedStatement.setDBDefault o la constante DB2PreparedStatement.DB_PARAMETER_DEFAULT simplifica la programación de la operación INSERT. Si no se utiliza DB2PreparedStatement.setDBDefault o DB2PreparedStatement.DB_PARAMETER_DEFAULT, serán necesarios hasta 32 objetos PreparedStatement distintos para cubrir todas las combinaciones de valores de entrada que sean por omisión y que no sean por omisión.

Uso de métodos DB2ResultSet o constantes DB2PreparedStatement para proporcionar información sobre parámetros ampliados

Utilice métodos DB2ResultSet o ResultSet con constantes DB2PreparedStatement para asignar valores por omisión a columnas de destino en un DB2ResultSet.

Acerca de esta tarea

Siga estos pasos para actualizar un ResultSet con información de cliente ampliada.

Procedimiento

1. Cree un objeto PreparedStatement.

   La sentencia de SQL es una sentencia SELECT.
2. Invoque métodos PreparedStatement.setXXX para pasar valores a cualquier parámetro de entrada.
3. Invoque el método PreparedStatement.executeQuery para obtener la tabla de resultados de la sentencia SELECT en un objeto ResultSet.
4. Coloque el cursor en la fila que desea actualizar o insertar.
5. Actualice columnas en la fila ResultSet.
   - Si no utiliza updateObject para actualizar un valor:
     - Para asignar el valor por omisión a la columna de destino del ResultSet, convierta el ResultSet a un DB2ResultSet e invoque DB2ResultSet.updateDBDefault.
   - Si utiliza updateObject para asignar el valor:
     - Para asignar el valor por omisión a la columna de destino del ResultSet, invoque ResultSet.updateObject con DB2PreparedStatement.DB_PARAMETER_DEFAULT como valor asignado.
6. Ejecute ResultSet.updateRow si está actualizando una fila existente o ResultSet.insertRow si inserta una fila nueva.

Ejemplo

El código siguiente inserta una fila en un ResultSet con el valor por omisión en la segunda columna y no modifica el valor en la primera columna. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
import java.sql.*;
import com.ibm.db2.jcc.*;

Connection conn;
...
PreparedStatement p = conn.prepareStatement (          1
  "SELECT MGRNO, LOCATION " +
  "FROM DEPARTMENT");
ResultSet rs = p.executeQuery ();                      3
rs.next();
rs.moveToInsertRow();                                  4
((DB2ResultSet)rs).updateDBDefault (2);                5
rs.insertRow();                                        6
...
rs.close();                          // Cerrar ResultSet
p.close();                           // Cerrar PreparedStatement
```

El código siguiente utiliza la interfaz ResultSet con las constantes DB2PreparedStatement para llevar a cabo la misma función que en el ejemplo anterior. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
import java.sql.*;
import com.ibm.db2.jcc.*;

Connection conn;
...
PreparedStatement p = conn.prepareStatement (          1
  "SELECT MGRNO, LOCATION " +
  "FROM DEPARTMENT");
ResultSet rs = p.executeQuery ();                      3
rs.next();
rs.moveToInsertRow();                                  4
rs.updateObject (2,                                    5
 DB2PreparedStatement.DB_PARAMETER_DEFAULT);
rs.insertRow();                                        6
...
rs.close();                            // Cerrar ResultSet
p.close();                             // Cerrar PreparedStatement
```

Bloqueo optimista en aplicaciones JDBC

Puede escribir aplicaciones JDBC para sacar provecho del bloqueo optimista sobre una fuente de datos.

El *bloqueo optimista* es una técnica que las aplicaciones pueden utilizar para liberar bloqueos entre operaciones SELECT y UPDATE o DELETE. Si las filas seleccionadas cambian antes de que la aplicación realice la actualización o supresión, la operación UPDATE o DELETE falla. El bloqueo optimista minimiza el tiempo durante el cual un recurso determinado no puede ser utilizado por otras transacciones.

Para conexiones con una fuente de datos DB2 para i, la utilización del bloqueo optimista requiere DB2 para i V6R1 o posterior.

En general, una aplicación sigue estos pasos para utilizar el bloqueo optimista:

1. Selecciona filas de una tabla.
2. Libera bloqueos mantenidos en la tabla.
3. Actualiza las filas seleccionadas, si no han cambiado.

Para comprobar si la fila ha cambiado, la aplicación consulta el símbolo de cambio de fila. El símbolo de cambio de fila no es siempre un indicador exacto de si una fila ha cambiado. Si crea una tabla con una columna de indicación de fecha y hora de cambio de fila, el símbolo de cambio de fila es totalmente exacto. Si crea la tabla sin una columna de indicación de fecha y hora de cambio de fila, o modifica una tabla para añadir una columna de indicación de fecha y hora de cambio de fila, el símbolo de cambio de fila puede que no refleje fielmente las actualizaciones hechas en una fila. Esto significa que el símbolo de cambio de fila puede indicar que una fila ha cambiado aunque no sea así. Esta condición se denomina condición de *negativo falso*.

Cuando escriba una aplicación JDBC para realizar un bloqueo optimista, seguirá pasos similares:

1. Prepare y ejecute una consulta.

Indique si desea información sobre el bloqueo optimista y si esa información puede incluir negativos falsos.

2. Determine si el ResultSet tiene información sobre el bloqueo optimista y si esa información puede producir negativos falsos.

   Basándose en el tipo de información sobre el bloqueo optimista, puede decidir si desea continuar utilizando el bloqueo optimista.

3. Libera bloqueos mantenidos en la tabla.
4. Actualice las filas seleccionadas, si el símbolo de cambio de fila indica que las filas no han cambiado.

El código siguiente muestra cómo una aplicación JDBC puede realizar el bloqueo optimista. Los números contenidos en el ejemplo corresponden a los pasos listados anteriormente.

```
com.ibm.db2.jcc.DB2Statement s1 =
  (com.ibm.db2.jcc.DB2Statement)conn.createStatement();
ResultSet rs =
 ((com.ibm.db2.jcc.DB2Statement)s1).executeDB2OptimisticLockingQuery
 ("SELECT EMPNO, SALARY FROM EMP WHERE EMP.LASTNAME = 'HAAS'",
  com.ibm.db2.jcc.DB2Statement.RETURN_OPTLOCK_COLUMN_NO_FALSE_NEGATIVES); 1
                                  // Indica que desea realizar bloqueo
                                  // optimista, y que desea información
                                  // sobre bloqueo optimista que
                                  // no produzca negativos falsos
ResultSetMetaData rsmd = rs.getMetaData();
int optColumns =                                                          2
  ((com.ibm.db2.jcc.DB2ResultSetMetaData)rsmd).getDB2OptimisticLockingColumns();
                                  // Recupere la información sobre
                                  // el bloqueo optimista.
boolean optColumnsReturned = false;

if (optColumns == 0);             // Si no se devuelve información sobre
                                  // bloqueo optimista, no intente realizar
                                  // bloqueo optimista.
else if (optColumns == 1);        // El valor 1 no se devuelve nunca si se especifica
                                  // RETURN_OPTLOCK_COLUMN_NO_FALSE_NEGATIVES,
                                  // pues 1 indica que podría haber negativos
                                  // falsos.
else if (optColumns == 2)         // Si se devuelve información sobre bloqueo
   optColumnsReturned = true;     // optimista y no aparecerán negativos falsos,
                                  // intente utilizar el bloqueo optimista.

rs.next();                        // Recupere el contenido de ResultSet
int emp_id = rs.getInt(1);
double salary = rs.getDouble(2);

long rowChangeToken = 0;
Object rid = null;
int type = -1;

if (optColumnsReturned) {
  rowChangeToken =                // Obtenga el símbolo de cambio de fila.
    ((com.ibm.db2.jcc.DB2ResultSet)rs).getDB2RowChangeToken();
  rid = ((com.ibm.db2.jcc.DB2ResultSet)rs).getDB2RID();
                                  // Obtenga RID, que sirve para
                                  // identificar la fila de forma exclusiva.
  int type = ((com.ibm.db2.jcc.DB2ResultSet)rs).getDB2RIDType ();
                                  // Obtenga el tipo de datos del RID.
}
// ************************************************
// Libere los bloqueos o desconecte de la base de datos.
// Realice algunas tareas sobre los datos recuperados.
// Vuelva a conectar con la fuente de datos.
// ************************************************
...
PreparedStatement s2 =
  conn.prepareStatement ("UPDATE EMP SET SALARY = ? " +
  "WHERE EMPNO = ? AND ROW CHANGE TOKEN FOR EMP = ? and " +
  "RID_BIT(EMP) = ?");
                                  // Sentencia para actualizar las
                                  // filas seleccionadas previamente
                                  // que no han cambiado.
s2.setDouble(1, salary+10000);
s2.setInt(2, emp_id);
```

```
                                  // Establezca los nuevos valores de la fila.
s2.setLong(3, rowChangeToken);
                                  // Establezca el símbolo de cambio de fila
                                  // de la fila recuperada anteriormente.
if (type == java.sql.Types.BIGINT)
  s2.setLong (4, ((Long)rid).longValue());
else if (type == java.sql.Types.VARBINARY)
  s2.setBytes (4, (byte[])rid);
                                  // Establezca el RID de la fila
                                  // recuperada anteriormente.
                                  // Utilice el método setXXX correcto
                                  // para el tipo de datos del RID.
int updateCount = s2.executeUpdate();                                    3
                                  // Realice la actualización.
if (updateCount == 1);            // La actualización ha sido satisfactoria.
else                              // La actualización ha fallado.
...
```

SQL compuesto en aplicaciones Java

Las cláusulas de las sentencias SQLJ en las aplicaciones SQLJ o las sentencias de SQL en las aplicaciones JDBC incluyen sentencias compuestas.

Una sentencia compuesta es un bloque BEGIN-END que incluye sentencias de SQL y de procedimiento. Todas las sentencias compuestas de aplicaciones JDBC y SQLJ se ejecutan dinámicamente.

El servidor de datos debe ser DB2 Database para Linux, UNIX y Windows Versión 9.7 o posterior.

IBM Data Server Driver para JDBC y SQLJ pasa el SQL compuesto al servidor de datos sin modificarlo.

Si el SQL compuesto contiene marcadores de parámetro, tendrá que establecer la propiedad enableNamedParameterMarkers de Connection o DataSource en DB2BaseDataSource.YES (1).

Además, en el servidor de datos tendrá que establecer la variable de registro DB2_COMPATIBILITY_VECTOR para habilitar la compilación y ejecución de PL/SQL.

El ejemplo siguiente muestra código JDBC que ejecuta una sentencia de SQL compuesto en línea. La sentencia de SQL compuesto contiene marcadores de parámetro, por lo que tendrá que establecer la propiedad enableNamedParameterMarkers en DB2BaseDataSource.YES (1).

```
...
Properties properties = new Properties(); // Crear un objeto Properties
properties.put("user", "db2adm");         // Establecer ID usuario para la conexión
properties.put("password", "db2adm");     // Establecer contraseña para la conexión
properties.put("enableNamedParameterMarkers",
  new String("" +
  com.ibm.db2.jcc.DB2BaseDataSource.YES + ""));
                                          // Establecer enableNamedParameterMarkers
String url = "jdbc:db2://luw1.myloc.ibm.com:9896/sample";
                                          // Establecer URL para la fuente de datos
Connection conn1 = DriverManager.getConnection(url, properties);
                                          // Crear la conexión
...
                                          // Preparar y ejecutar la sentencia
                                          // compuesta tal y como se prepara
                                          // una única sentencia de SQL
PreparedStatement ps = conn1.prepareStatement(
     "BEGIN atomic " +
       " for row as " +
```

```
      "  select pk, c1, discretize (c1) as d from source " +
      " do " +
      "  if row.d is null then " +
      "   insert into except values (row.pk, ?); " +
      "  else " +
      "   insert into target values (row.pk, ?); " +
      "  end if; " +
      " end for; " +
    "end");
  ps.setInt(1, 98);
  ps.setInt(2, 88);
  ps.execute();
...
```

Datos XML en aplicaciones JDBC

En las aplicaciones JDBC, puede almacenar en columnas XML y recuperar datos de columnas XML.

En las tablas de base de datos, se utiliza el tipo de datos incorporado XML para almacenar datos XML en una columna en forma de conjunto estructurado de nodos en formato de árbol.

Las aplicaciones JDBC pueden enviar datos XML al servidor de datos o recuperar datos XML del servidor de datos de una de estas maneras:

- Como datos XML textuales
- Como datos XML binarios, si el servidor de datos los admite

En las aplicaciones JDBC, puede:

- Almacenar un documento XML completo en una columna XML mediante los métodos setXXX.
- Recuperar un documento XML completo de una columna XML mediante los métodos getXXX.
- Recuperar una secuencia de un documento en una columna XML mediante la función XMLQUERY de SQL para recuperar la secuencia en una secuencia serializada en la base de datos y, a continuación, utilizar métodos getXXX para recuperar los datos en una variable de aplicación.
- Recuperar una secuencia de un documento en una columna XML mediante una expresión XQuery, a la que se añade la serie 'XQUERY' como prefijo, para recuperar los elementos de la secuencia en una tabla de resultados de la base de datos. En dicha base de datos, cada fila de la tabla de resultados representa un elemento de la secuencia. A continuación, se utilizan los métodos getXXX para recuperar los datos en variables de la aplicación.
- Recuperar una secuencia de un documento en una columna XML en forma de una tabla definida por el usuario mediante la función XMLTABLE de SQL para definir la tabla de resultados y recuperarla. A continuación, se utilizan los métodos getXXX para recuperar los datos de la tabla de resultados en variables de la aplicación.

Se pueden utilizar objetos `java.sql.SQLXML` de JDBC 4.0 para recuperar y actualizar datos de columnas XML. Las invocaciones de métodos de metadatos, tales como ResultSetMetaData.getColumnTypeName, devuelven el valor entero `java.sql.Types.SQLXML` para una columna de tipo XML.

Actualizaciones de columnas XML en aplicaciones JDBC

En una aplicación JDBC, se pueden actualizar o insertar datos en columnas XML de una tabla situada en un servidor de datos DB2 utilizando datos textuales XML. Se pueden actualizar o insertar datos en columnas XML de una tabla usando datos XML binarios (datos que están en el formato Extensible Dynamic Binary XML DB2 Client/Server Binary XML) si el servidor de datos da soporte a los datos XML binarios.

La tabla siguiente lista los métodos y los correspondientes tipos de datos de entrada que puede utilizar para insertar datos en columnas XML.

Tabla 20. Métodos y tipos de datos para actualizar columnas XML

Método	Tipo de datos de entrada
PreparedStatement.setAsciiStream	InputStream
PreparedStatement.setBinaryStream	InputStream
PreparedStatement.setBlob	Blob
PreparedStatement.setBytes	byte[]
PreparedStatement.setCharacterStream	Reader
PreparedStatement.setClob	Clob
PreparedStatement.setObject	byte[], Blob, Clob, SQLXML, DB2Xml (en desuso), InputStream, Reader, String
PreparedStatement.setSQLXML[1]	SQLXML
PreparedStatement.setString	Serie

Nota:

1. Este método requiere JDBC 4.0 o posterior.

La codificación de los datos XML se puede obtener a partir de los propios datos, lo cual se conoce como datos *codificados internamente*, o a partir de fuentes externas, lo cual se conoce como datos *codificados externamente*. Los datos XML que se envían al servidor de bases de datos como datos binarios se tratan como datos codificados internamente. Los datos XML que se envían a la fuente de datos como datos de tipo carácter se tratan como datos codificados externamente.

La codificación externa utilizada para aplicaciones Java es siempre la codificación Unicode.

Los datos codificados externamente pueden tener una codificación interna. Es decir, los datos se pueden enviar a la fuente de datos como datos de tipo carácter, pero los datos contienen información de codificación. La fuente de datos trata las incompatibilidades entre la codificación interna y la codificación externa de la manera siguiente:

- Si la fuente de datos es DB2 Database para Linux, UNIX y Windows, la fuente de base de datos genera un error si las codificaciones externa e interna son incompatibles, a menos que ambas codificaciones sean Unicode. Si las codificaciones externa e interna son Unicode, la fuente de base de datos no tiene en cuenta la codificación interna.
- Si la fuente de base de datos es DB2 para z/OS, la fuente de base de datos no tiene en cuenta la codificación interna.

Los datos de las columnas XML se almacenan utilizando la codificación UTF-8. La fuente de base de datos maneja la conversión de los datos desde su codificación interna o externa a la codificación UTF-8.

Ejemplo: el ejemplo siguiente muestra la inserción de datos de un objeto SQLXML en una columna XML. Los datos son de tipo carácter (String), por lo que la fuente de base de datos trata los datos como codificados externamente.

```
public void insertSQLXML()
{
 Connection con = DriverManager.getConnection(url);
  SQLXML info = con.createSQLXML();
                            // Crear objeto SQLXML
  PreparedStatement insertStmt = null;
  String infoData =
    "<customerinfo xmlns=""http://posample.org"" " +
    "Cid=""1000"">...</customerinfo>";
  info.setString(infoData);
                            // Llenar objeto SQLXML con datos
  int cid = 1000;
  try {
    sqls = "INSERT INTO CUSTOMER (CID, INFO) VALUES (?, ?)";
    insertStmt = con.prepareStatement(sqls);
    insertStmt.setInt(1, cid);
    insertStmt.setSQLXML(2, info);
                            // Asignar el valor del objeto SQLXML
                           // a un parámetro de entrada
    if (insertStmt.executeUpdate() != 1) {
      System.out.println("insertSQLXML: Ningún registro insertado.");
    }
  }
  catch (IOException ioe) {
   ioe.printStackTrace();
  }
  catch (SQLException sqle) {
    System.out.println("insertSQLXML: Excepción de SQL: " +
      sqle.getMessage());
    System.out.println("insertSQLXML: Estado de SQL: " +
      sqle.getSQLState());
    System.out.println("insertSQLXML: Código de error de SQL: " +
      sqle.getErrorCode());

  }
}
```

Ejemplo: el ejemplo siguiente demuestra la inserción de datos de un archivo en una columna XML. Los datos se insertan como datos binarios, de manera que el servidor de bases de datos respeta la codificación interna.

```
public void insertBinStream(Connection conn)
{
  PreparedStatement insertStmt = null;
  String sqls = null;
  int cid = 0;
  Statement stmt=null;
  try {
    sqls = "INSERT INTO CUSTOMER (CID, INFO) VALUES (?, ?)";
    insertStmt = conn.prepareStatement(sqls);
    insertStmt.setInt(1, cid);
    File file = new File(fn);
    insertStmt.setBinaryStream(2,
      new FileInputStream(file), (int)file.length());
    if (insertStmt.executeUpdate() != 1) {
      System.out.println("insertBinStream: Ningún registro insertado.");
    }
  }
```

```
    catch (IOException ioe) {
     ioe.printStackTrace();
    }
    catch (SQLException sqle) {
      System.out.println("insertBinStream: SQL Exception: " +
        sqle.getMessage());
      System.out.println("insertBinStream: SQL State: " +
        sqle.getSQLState());
      System.out.println("insertBinStream: SQL Error Code: " +
        sqle.getErrorCode());

    }
  }
```

Ejemplo: el ejemplo siguiente demuestra la inserción de datos XML binarios de un archivo en una columna XML.

```
...
SQLXML info = conn.createSQLXML();
OutputStream os = info.setBinaryStream ();
FileInputStream fis = new FileInputStream("c7.xml");
int read;
while ((read = fis.read ()) != -1) {
  os.write (read);
}

PreparedStatement insertStmt = null;
String sqls = null;
int cid = 1015;
sqls = "INSERT INTO MyCustomer (Cid, Info) VALUES (?, ?)";
insertStmt = conn.prepareStatement(sqls);
insertStmt.setInt(1, cid);
insertStmt.setSQLXML(2, info);
insertStmt.executeUpdate();
```

Recuperación de datos XML en aplicaciones JDBC

En las aplicaciones JDBC, se utilizan los métodos ResultSet.getXXX o ResultSet.getObject para recuperar datos de columnas XML.

En una aplicación JDBC, se pueden recuperar datos de columnas XML en una tabla de DB2 como datos textuales XML. Se pueden recuperar datos de las columnas XML de una tabla como datos XML binarios (datos que están en el formato Extensible Dynamic Binary XML DB2 Client/Server Binary XML) si el servidor de datos da soporte a los datos XML binarios.

Puede utilizar una de las técnicas siguientes para recuperar datos XML:

- Utilice el método ResultSet.getSQLXML para recuperar los datos. Luego utilice un método SQLXML.getXXX para transformar los datos a un tipo de datos de salida compatible. Esta técnica requiere JDBC 4.0 o posterior.

  Por ejemplo, se pueden recuperar datos mediante el método SQLXML.getBinaryStream o el método SQLXML.getSource.
- Utilice un método ResultSet.getXXX que no sea ResultSet.getObject para recuperar los datos en un tipo que sea compatible.
- Utilice el método ResultSet.getObject para recuperar los datos y, a continuación, conviértalos al tipo DB2Xml y asígnelos a un objeto DB2Xml. A continuación, utilice un método DB2Xml.getDB2XXX o DB2Xml.getDB2XmlXXX para recuperar los datos en un tipo de datos de salida compatible.

  Es necesario utilizar esta técnica si no utiliza una versión de IBM Data Server Driver para JDBC y SQLJ que soporte JDBC 4.0.

La tabla siguiente muestra los métodos ResultSet y sus correspondientes tipos de datos de salida para recuperar datos XML.

Tabla 21. Métodos ResultSet y tipos de datos para recuperar datos XML

Método	Tipo de datos de salida
ResultSet.getAsciiStream	InputStream
ResultSet.getBinaryStream	InputStream
ResultSet.getBytes	byte[]
ResultSet.getCharacterStream	Reader
ResultSet.getObject	Object
ResultSet.getSQLXML	SQLXML
ResultSet.getString	Serie

La tabla siguiente muestra los métodos que puede invocar para recuperar datos de un objeto java.sql.SQLXML o com.ibm.db2.jcc.DB2Xml, los tipos de datos de salida correspondientes y el tipo de codificación utilizado en las declaraciones XML.

Tabla 22. Métodos SQLXML y DB2Xml, tipos de datos y especificaciones de codificación añadidas

Método	Tipo de datos de salida	Tipo de declaración de codificación interna XML añadida
SQLXML.getBinaryStream	InputStream	Ninguno
SQLXML.getCharacterStream	Reader	Ninguno
SQLXML.getSource	Source[1]	Ninguno
SQLXML.getString	Serie	Ninguno
DB2Xml.getDB2AsciiStream	InputStream	Ninguno
DB2Xml.getDB2BinaryStream	InputStream	Ninguno
DB2Xml.getDB2Bytes	byte[]	Ninguno
DB2Xml.getDB2CharacterStream	Reader	Ninguno
DB2Xml.getDB2String	Serie	Ninguno
DB2Xml.getDB2XmlAsciiStream	InputStream	US-ASCII
DB2Xml.getDB2XmlBinaryStream	InputStream	Especificado por el parámetro getDB2XmlBinaryStream *codificaciónDestino*
DB2Xml.getDB2XmlBytes	byte[]	Especificado por el parámetro DB2Xml.getDB2XmlBytes *codificaciónDestino*
DB2Xml.getDB2XmlCharacterStream	Reader	ISO-10646-UCS-2
DB2Xml.getDB2XmlString	Serie	ISO-10646-UCS-2

Nota:

1. El invocador de getSource especifica la clase que se devuelve, pero la clase debe ampliar javax.xml.transform.Source.

Si la aplicación ejecuta la función XMLSERIALIZE en los datos que se deben devolver, después de la ejecución de la función, los datos tendrán el tipo especificado en la función XMLSERIALIZE, no el tipo de datos XML. Por consiguiente, el controlador maneja los datos como el tipo especificado y pasa por alto cualquier declaración de codificación interna.

Ejemplo: el ejemplo siguiente muestra la recuperación de datos de una columna XML para colocarlos en un objeto SQLXML, y luego la utilización del método SQLXML.getString para recuperar los datos y colocarlos en una serie de caracteres.

```
public void fetchToSQLXML(long cid, java.sql.Connection conn)
  {
    System.out.println(">> fetchToSQLXML: Get XML data as an SQLXML object " +
    "using getSQLXML");
    PreparedStatement selectStmt = null;
    String sqls = null, stringDoc = null;
    ResultSet rs = null;

    try{
      sqls = "SELECT info FROM customer WHERE cid = " + cid;
      selectStmt = conn.prepareStatement(sqls);
      rs = selectStmt.executeQuery();

      // Obtener metadatos
      // El tipo de la columna XML es el entero java.sql.Types.OTHER
      ResultSetMetaData meta = rs.getMetaData();
      int colType = meta.getColumnType(1);
      System.out.println("fetchToSQLXML: Column type = " + colType);
      while (rs.next()) {
        // Recuperar los datos XML con getSQLXML.
        // A continuación escribirlo en una serie de texto con
        // codificación ISO-10646-UCS-2 interna explícita.
        java.sql.SQLXML xml = rs.getSQLXML(1);
        System.out.println (xml.getString());
      }
      rs.close();
    }
    catch (SQLException sqle) {
      System.out.println("fetchToSQLXML: SQL Exception: " +
        sqle.getMessage());
      System.out.println("fetchToSQLXML: SQL State: " +
        sqle.getSQLState());
      System.out.println("fetchToSQLXML: SQL Error Code: " +
        sqle.getErrorCode());
    }
  }
```

Ejemplo: el ejemplo siguiente muestra la recuperación de datos de una columna XML para colocarlos en un objeto SQLXML, y luego la utilización del método SQLXML.getBinaryStream para recuperar los datos como datos binarios en una corriente de entrada.

```
String sql = "SELECT INFO FROM Customer WHERE Cid='1000'";
PreparedStatement pstmt = con.prepareStatement(sql);
ResultSet resultSet = pstmt.executeQuery();
// Obtener el resultado XML como corriente binaria
SQLXML sqlxml = resultSet.getSQLXML(1);
InputStream binaryStream = sqlxml.getBinaryStream();
```

Ejemplo: en el ejemplo siguiente, se muestra la recuperación de datos desde una columna XML a una variable de tipo String.

```
public void fetchToString(long cid, java.sql.Connection conn)
  {
    System.out.println(">> fetchToString: Get XML data " +
    "using getString");
    PreparedStatement selectStmt = null;
    String sqls = null, stringDoc = null;
    ResultSet rs = null;

    try{
      sqls = "SELECT info FROM customer WHERE cid = " + cid;
      selectStmt = conn.prepareStatement(sqls);
```

```
      rs = selectStmt.executeQuery();

      // Obtener metadatos
      // El tipo de la columna XML es el entero java.sql.Types.OTHER
      ResultSetMetaData meta = rs.getMetaData();
      int colType = meta.getColumnType(1);
      System.out.println("fetchToString: Column type = " + colType);

      while (rs.next()) {
        stringDoc = rs.getString(1);
        System.out.println("Document contents:");
        System.out.println(stringDoc);
 }
    catch (SQLException sqle) {
      System.out.println("fetchToString: SQL Exception: " +
        sqle.getMessage());
      System.out.println("fetchToString: SQL State: " +
        sqle.getSQLState());
      System.out.println("fetchToString: SQL Error Code: " +
        sqle.getErrorCode());
    }
  }
```

Ejemplo: en el ejemplo siguiente, se muestra la recuperación de datos desde una columna XML a un objeto DB2Xml. A continuación, se utiliza el método DB2Xml.getDB2XmlString para recuperar datos en una serie con una declaración XML añadida con una especificación de codificación ISO-10646-UCS-2.

```
public void fetchToDB2Xml(long cid, java.sql.Connection conn)
  {
    System.out.println(">> fetchToDB2Xml: Get XML data as a DB2XML object " +
    "using getObject");
    PreparedStatement selectStmt = null;
    String sqls = null, stringDoc = null;
    ResultSet rs = null;

    try{
      sqls = "SELECT info FROM customer WHERE cid = " + cid;
      selectStmt = conn.prepareStatement(sqls);
      rs = selectStmt.executeQuery();

      // Obtener metadatos
      // El tipo de la columna XML es el entero java.sql.Types.OTHER
      ResultSetMetaData meta = rs.getMetaData();
      int colType = meta.getColumnType(1);
      System.out.println("fetchToDB2Xml: Column type = " + colType);
      while (rs.next()) {
        // Recupera los datos XML con getObject y convierte el objeto
        // como un objeto DB2Xml. Luego, lo escribe en una serie con
        // codificación ISO-10646-UCS-2 interna explícita.
        com.ibm.db2.jcc.DB2Xml xml =
          (com.ibm.db2.jcc.DB2Xml) rs.getObject(1);
        System.out.println (xml.getDB2XmlString());
      }
      rs.close();
    }
    catch (SQLException sqle) {
      System.out.println("fetchToDB2Xml: SQL Exception: " +
        sqle.getMessage());
      System.out.println("fetchToDB2Xml: SQL State: " +
        sqle.getSQLState());
      System.out.println("fetchToDB2Xml: SQL Error Code: " +
        sqle.getErrorCode());
    }
  }
```

Invocación de rutinas con parámetros XML en aplicaciones Java

Las aplicaciones Java pueden llamar a procedimientos almacenados en fuentes de datos de DB2 Database para Linux, UNIX y Windows o DB2 para z/OS que tengan parámetros XML.

Para los procedimientos de SQL nativos, los parámetros XML de la definición de procedimiento almacenado tienen el tipo XML. Para los procedimientos almacenados externos y las funciones definidas por el usuario en fuentes de datos DB2 Database para Linux, UNIX y Windows, los parámetros XML de la definición de rutina tienen el tipo XML AS CLOB. Cuando invoca un procedimiento almacenado o una función definida por el usuario que tiene parámetros XML, necesita utilizar un tipo de datos compatible en la sentencia de invocación.

Para invocar una rutina con parámetros de entrada XML desde un programa JDBC, utilice parámetros del tipo java.sql.SQLXML o com.ibm.db2.jcc.DB2Xml. Para registrar parámetros XML de salida, registre los parámetros como pertenecientes al tipo java.sql.Types.SQLXML o com.ibm.db2.jcc.DB2Types.XML. (Los tipos com.ibm.db2.jcc.DB2Xml y com.ibm.db2.jcc.DB2Types.XML están en desuso).

Ejemplo: El programa JDBC que llama un procedimiento almacenado que utiliza tres parámetros XML: un parámetro IN, un parámetro OUT y un parámetro INOUT. Este ejemplo requiere JDBC 4.0 o posterior.

```
java.sql.SQLXML in_xml = xmlvar;
java.sql.SQLXML out_xml = null;
java.sql.SQLXML inout_xml = xmlvar;
                                  // Declarar un parámetro XML de entrada,
                                  // parámetro INOUT XML
Connection con;
CallableStatement cstmt;
ResultSet rs;
...
cstmt = con.prepareCall("CALL SP_xml(?,?,?)");
                                  // Crear un objeto CallableStatement
cstmt.setObject (1, in_xml);      // Definir parámetro de entrada
cstmt.setObject (3, inout_xml);   // Definir parámetro inout
cstmt.registerOutParameter (2, java.sql.Types.SQLXML);
                                  // Registrar parámetros de entrada y de salida
cstmt.registerOutParameter (3, java.sql.Types.SQLXML);
cstmt.executeUpdate();            // Invocar el procedimiento almacenado
out_xml = cstmt.getSQLXML(2);     // Obtener el valor del parámetro OUT
inout_xml = cstmt.getSQLXML(3);   // Obtener el valor del parámetro INOUT
System.out.println("Valores de parámetros de llamada a SP_xml: ");
System.out.println("Valor de parámetro de salida ");
MyUtilities.printString(out_xml.getString());
                          // Utilizar el método SQLXML.getString
                          // para convertir el valor out_xml
                          // a una serie para impresión.
                          // Llamar a un método definido por el usuario denominado
                          // printString (no se muestra) para imprimir
                          // el valor.
System.out.println("Valor de parámetro INOUT");
MyUtilities.printString(inout_xml.getString());
                          // Utilizar el método SQLXML.getString
                          // para convertir el valor inout_xml
                          // a una serie para impresión.
                          // Llamar a un método definido por el usuario denominado
                          // printString (no se muestra) para imprimir
                          // el valor.
```

Para invocar una rutina con parámetros XML desde un programa SQLJ, utilice parámetros del tipo java.sql.SQLXML o com.ibm.db2.jcc.DB2Xml.

Ejemplo: El programa SQLJ que llama un procedimiento almacenado que utiliza tres parámetros XML: un parámetro IN, un parámetro OUT y un parámetro INOUT. Este ejemplo requiere JDBC 4.0 o posterior.

```
java.sql.SQLXML in_xml = xmlvar;
java.sql.SQLXML out_xml = null;
java.sql.SQLXML inout_xml = xmlvar;
                                  // Declarar un parámetro XML de entrada,
                                  // parámetro INOUT XML
...
#sql [myConnCtx] {CALL SP_xml(:IN in_xml,
                              :OUT out_xml,
                              :INOUT inout_xml)};
                                  // Invocar el procedimiento almacenado
System.out.println("Valores de parámetros de llamada a SP_xml: ");
System.out.println("Valor de parámetro de salida ");
MyUtilities.printString(out_xml.getString());
                             // Utilizar el método SQLXML.getString
                             // para convertir el valor out_xml
                             // a una serie para impresión.
                             // Llamar a un método definido por el usuario denominado
                             // printString (no se muestra) para imprimir
                             // el valor.
System.out.println("Valor de parámetro INOUT");
MyUtilities.printString(inout_xml.getString());
                             // Utilizar el método SQLXML.getString
                             // para convertir el valor inout_xml
                             // a una serie para impresión.
                             // Llamar a un método definido por el usuario denominado
                             // printString (no se muestra) para imprimir
                             // el valor.
```

Soporte de Java para el registro y la eliminación de esquemas XML

IBM Data Server Driver para JDBC y SQLJ proporciona métodos que le permiten escribir programas de aplicación Java para registrar y eliminar esquemas XML y sus componentes.

Los métodos son:

DB2Connection.registerDB2XMLSchema
Registra un esquema XML en DB2, utilizando uno o más documentos de esquema XML. Existen dos modalidades de este método: una modalidad para documentos de esquema XML que proceden de objetos InputStream, y una modalidad para documentos de esquema XML que residen en un `String`.

DB2Connection.deregisterDB2XMLObject
Elimina una definición de esquema XML en DB2.

DB2Connection.updateDB2XmlSchema
Sustituye los documentos de esquema XML contenidos en un esquema XML registrado por los documentos de esquema XML de otro esquema XML registrado. Opcionalmente, descarta el esquema XML cuyo contenido se copia. Este método está disponible solamente para conexiones con DB2 Database para Linux, UNIX y Windows.

Antes de poder invocar estos métodos, se deben instalar los procedimientos almacenados que dan soporte a estos métodos en el servidor de bases de datos DB2.

Ejemplo - Registro de un esquema XML: el ejemplo siguiente muestra la utilización de registerDB2XmlSchema para registrar un esquema XML en DB2 utilizando un solo documento de esquema XML (customer.xsd) que se lee a partir de una corriente de datos de entrada. El nombre de esquema SQL correspondiente al esquema registrado es SYSXSR. No se registran propiedades adicionales.

```
public static void registerSchema(
  Connection con,
  String schemaName)
  throws SQLException {
  // Definir parámetros de registerDB2XmlSchema
  String[] xmlSchemaNameQualifiers = new String[1];
  String[] xmlSchemaNames = new String[1];
  String[] xmlSchemaLocations = new String[1];
  InputStream[] xmlSchemaDocuments = new InputStream[1];
  int[] xmlSchemaDocumentsLengths = new int[1];
  java.io.InputStream[] xmlSchemaDocumentsProperties = new InputStream[1];
  int[] xmlSchemaDocumentsPropertiesLengths = new int[1];
  InputStream xmlSchemaProperties;
  int xmlSchemaPropertiesLength;
  //Establecer valores de parámetros
  xmlSchemaLocations[0] = "";
  FileInputStream fi = null;
  xmlSchemaNameQualifiers[0] = "SYSXSR";
  xmlSchemaNames[0] = schemaName;
  try {
    fi = new FileInputStream("customer.xsd");
    xmlSchemaDocuments[0] = new BufferedInputStream(fi);
  } catch (FileNotFoundException e) {
         e.printStackTrace();
  }
  try {
     xmlSchemaDocumentsLengths[0] = (int) fi.getChannel().size();
     System.out.println(xmlSchemaDocumentsLengths[0]);
  } catch (IOException e1) {
         e1.printStackTrace();
  }
  xmlSchemaDocumentsProperties[0] = null;
  xmlSchemaDocumentsPropertiesLengths[0] = 0;
  xmlSchemaProperties = null;
  xmlSchemaPropertiesLength = 0;
  DB2Connection ds = (DB2Connection) con;
  // Invocar registerDB2XmlSchema
  ds.registerDB2XmlSchema(
    xmlSchemaNameQualifiers,
    xmlSchemaNames,
    xmlSchemaLocations,
    xmlSchemaDocuments,
    xmlSchemaDocumentsLengths,
    xmlSchemaDocumentsProperties,
    xmlSchemaDocumentsPropertiesLengths,
    xmlSchemaProperties,
    xmlSchemaPropertiesLength,
    false);
}
```

Ejemplo - Eliminación de un esquema XML: el ejemplo siguiente muestra la utilización de deregisterDB2XmlObject para eliminar un esquema XML en DB2. El nombre de esquema SQL correspondiente al esquema registrado es SYSXSR.

```
public static void deregisterSchema(
  Connection con,
  String schemaName)
  throws SQLException {
  // Definir y asignar valores a parámetros de deregisterDB2XmlObject
  String xmlSchemaNameQualifier = "SYSXSR";
  String xmlSchemaName = schemaName;
```

```
  DB2Connection ds = (DB2Connection) con;
  // Invocar deregisterDB2XmlObject
  ds.deregisterDB2XmlObject(
    xmlSchemaNameQualifier,
    xmlSchemaName);
}
```

Ejemplo: Actualización de un esquema XML: el ejemplo siguiente se aplica solamente a las conexiones con DB2 Database para Linux, UNIX y Windows. Demuestra la utilización de updateDB2XmlSchema para actualizar el contenido de un esquema XML con el contenido de otro esquema XML. El esquema que se copia se mantiene en el depósito. El nombre de esquema SQL correspondiente a ambos esquemas registrados es SYSXSR.

```
public static void updateSchema(
  Connection con,
  String schemaNameTarget,
  String schemaNameSource)
  throws SQLException {
  // Definir y asignar valores a los parámetros de updateDB2XmlSchema
  String xmlSchemaNameQualifierTarget = "SYSXSR";
  String xmlSchemaNameQualifierSource = "SYSXSR";
  String xmlSchemaNameTarget = schemaNameTarget;
  String xmlSchemaNameSource = schemaNameSource;
  boolean dropSourceSchema = false;
  DB2Connection ds = (DB2Connection) con;
  // Invocar updateDB2XmlSchema
  ds.updateDB2XmlSchema(
    xmlSchemaNameQualifierTarget,
    xmlSchemaNameTarget,
  xmlSchemaNameQualifierSource,
  xmlSchemaNameSource,
    dropSourceSchema);
}
```

Control de transacciones en aplicaciones JDBC

En las aplicaciones JDBC, al igual que en otros tipos de aplicaciones SQL, el control de transacciones supone la confirmación y retrotracción explícita o implícita de transacciones, y definir el nivel de aislamiento de las transacciones.

Niveles de aislamiento de IBM Data Server Driver para JDBC y SQLJ

IBM Data Server Driver para JDBC y SQLJ es compatible con varios niveles de aislamiento, que corresponden a niveles de aislamiento del servidor de bases de datos.

Los niveles de aislamiento de JDBC se pueden definir para una unidad de trabajo dentro del programa JDBC, utilizando el método Connection.setTransactionIsolation. El nivel de aislamiento por omisión se puede establecer mediante la propiedad defaultIsolationLevel.

La tabla siguiente muestra los valores de *nivel* que puede especificar en el método Connection.setTransactionIsolation y sus equivalentes para el servidor de bases de datos DB2.

Tabla 23. Niveles de aislamiento equivalentes para JDBC y DB2

Valor para JDBC	Nivel de aislamiento de DB2
java.sql.Connection.TRANSACTION_SERIALIZABLE	Lectura repetible

Tabla 23. Niveles de aislamiento equivalentes para JDBC y DB2 (continuación)

Valor para JDBC	Nivel de aislamiento de DB2
java.sql.Connection.TRANSACTION_REPEATABLE_READ	Estabilidad de lectura
java.sql.Connection.TRANSACTION_READ_COMMITTED	Estabilidad del cursor
java.sql.Connection.TRANSACTION_READ_UNCOMMITTED	Lectura no confirmada

La tabla siguiente muestra los valores de *nivel* que puede especificar en el método Connection.setTransactionIsolation y sus equivalentes para IBM Informix.

Tabla 24. Niveles de aislamiento equivalentes en JDBC e IBM Informix

Valor para JDBC	Nivel de aislamiento en IBM Informix
java.sql.Connection.TRANSACTION_SERIALIZABLE	Lectura repetible
java.sql.Connection.TRANSACTION_REPEATABLE_READ	Lectura repetible
java.sql.Connection.TRANSACTION_READ_COMMITTED	Lectura confirmada
java.sql.Connection.TRANSACTION_READ_UNCOMMITTED	Lectura sucia
com.ibm.db2.jcc.DB2Connection.TRANSACTION_IDS_CURSOR_STABILITY	Estabilidad del cursor de IBM Informix
com.ibm.db2.jcc.DB2Connection.TRANSACTION_IDS_LAST_COMMITTED	Lectura confirmada, última confirmada

Confirmación o retrotracción de transacciones JDBC

En JDBC, para confirmar o retrotraer explícitamente transacciones, utilice los métodos commit o rollback.

Acerca de esta tarea

Por ejemplo:

```
 Connection con;
 ...
con.commit();
```

Si la modalidad de confirmación automática (autocommit) está activada, el gestor de bases de datos ejecuta una operación de confirmación después de la ejecución de cada sentencia de SQL. Para activar la modalidad de confirmación automática, invoque el método Connection.setAutoCommit(true). Para desactivar la modalidad de confirmación automática, invoque el método Connection.setAutoCommit(false). Para determinar si la modalidad de confirmación automática está activa, invoque el método Connection.getAutoCommit.

Las conexiones que intervienen en transacciones distribuidas no pueden invocar el método setAutoCommit(true).

Cuando el usuario cambia el estado de la confirmación automática, el gestor de bases de datos ejecuta una operación de confirmación si la aplicación no se encuentra ya en un límite de transacción.

Mientras una conexión participa en una transacción distribuida, la aplicación asociada no puede emitir los métodos commit o rollback.

Modalidades de confirmación automática por omisión de JDBC

La modalidad de confirmación automática depende de la fuente de datos a la que se conecta la aplicación JDBC.

Valor por omisión de la confirmación automática para fuentes de datos DB2

Para las conexiones con fuentes de datos DB2, la modalidad de confirmación automática por omisión es `true`.

Valor por omisión de la confirmación automática para fuentes de datos IBM Informix

Para las conexiones con fuentes de datos IBM Informix, la modalidad de confirmación automática por omisión depende del tipo de la fuente de datos. La tabla siguiente muestra los valores por omisión.

Tabla 25. Modalidades de confirmación automática por omisión para fuentes de datos IBM Informix

Tipo de fuente de datos	Modalidad de confirmación automática para transacciones locales	Modalidad de confirmación automática para transacciones globales
Base de datos compatible con ANSI	true	false
Base de datos no compatible con ANSI sin anotación cronológica	false	No aplicable
Base de datos no compatible con ANSI con anotación cronológica	true	false

Excepciones y avisos cuando se utiliza IBM Data Server Driver para JDBC y SQLJ

En las aplicaciones JDBC, los errores de SQL emiten excepciones, que el usuario trata mediante bloques `try/catch`. Los avisos de SQL no emiten excepciones, por lo que después de ejecutar sentencias de SQL es necesario invocar métodos para determinar si se han producido avisos.

IBM Data Server Driver para JDBC y SQLJ proporciona las clases e interfaces siguientes, las cuales proporcionan información sobre errores y avisos.

SQLException

La clase SQLException para el manejo de errores. Todos los métodos de JDBC emiten una instancia de SQLException cuando se produce un error durante la ejecución del método. De acuerdo con la especificación JDBC, un objeto SQLException contiene la información siguiente:

- Un valor `int` que contiene un código de error. SQLException.getErrorCode recupera este valor.
- Un objeto String que contiene el SQLSTATE o el valor nulo. SQLException.getSQLState recupera este valor.
- Un objeto String que contiene una descripción del error o el valor nulo. SQLException.getMessage recupera este valor.

- Un puntero que indica la ubicación del objeto SQLException siguiente o un valor nulo. SQLException.getNextException recupera este valor.

Cuando un método JDBC emite una SQLException, esa SQLException puede ser debida a la excepción Java subyacente que se produjo cuando IBM Data Server Driver para JDBC y SQLJ procesó el método. En este caso, la SQLException engloba la excepción subyacente, y se puede utilizar el método SQLException.getCause para obtener información sobre el error.

DB2Diagnosable

La interfaz com.ibm.db2.jcc.DB2Diagnosable específica de IBM Data Server Driver para JDBC y SQLJ amplía la clase SQLException. La interfaz DB2Diagnosable le proporciona más información sobre los errores producidos al acceder a la fuente de datos. Si el controlador JDBC detecta un error, DB2Diagnosable le proporciona la misma información que la clase SQLException estándar. Sin embargo, si el servidor de bases de datos detecta el error, DB2Diagnosable añade los métodos siguientes, que proporcionan información adicional sobre el error:

getSqlca
Devuelve un objeto DB2Sqlca con la información siguiente:
- Un código de error de SQL
- Los valores SQLERRMC
- El valor SQLERRP
- Los valores SQLERRD
- Los valores SQLWARN
- El SQLSTATE

getThrowable
Devuelve el objeto java.lang.Throwable que causó la excepción SQLException o un valor nulo si ese objeto no existe.

printTrace
Imprime información de diagnóstico.

Subclases de SQLException

Si está utilizando JDBC 4.0 o versión posterior, puede obtener información más específica que la proporcionada por una SQLException capturando las clases de excepción siguientes:

- SQLNonTransientException

  Se emite una SQLNonTransientException cuando una operación de SQL que falló anteriormente no se ejecuta con éxito cuando se reintenta la operación, a menos que se emprenda alguna acción correctora. La clase SQLNonTransientException tiene estas subclases:
  - SQLFeatureNotSupportedException
  - SQLNonTransientConnectionException
  - SQLDataException
  - SQLIntegrityConstraintViolationException
  - SQLInvalidAuthorizationSpecException
  - SQLSyntaxException
- SQLTransientException

  Se emite una SQLTransientException cuando una operación de SQL que falló anteriormente podría ejecutarse con éxito al reintentar la operación, sin intervención de la aplicación. La conexión sigue siendo válida con se emite una SQLTransientException. La clase SQLTransientException tiene estas subclases:

  - SQLTransientConnectionException
  - SQLTransientRollbackException
  - SQLTimeoutException
- SQLRecoverableException

  Se emite una SQLRecoverableException cuando una operación que falló anteriormente podría ejecutarse con éxito si la aplicación realiza algunos pasos de recuperación y reintenta la transacción. La conexión ya no es válida después de emitirse una SQLRecoverableException.
- SQLClientInfoException

  El método Connection.setClientInfo emite una SQLClientInfoException cuando no se pueden establecer una o más propiedades del cliente. La SQLClientInfoException indica qué propiedades no se pueden establecer.

BatchUpdateException

Un objeto BatchUpdateException contiene los elementos siguientes sobre un error producido al ejecutar un lote sentencias de SQL:
- Un objeto String que contiene una descripción del error, o bien `null`.
- Un objeto String que contiene el estado de SQL (SQLSTATE) de la sentencia de SQL anómala, o el valor `null`
- Un valor entero que contiene el código de error o cero
- Una matriz entera de cuentas de actualización para las sentencias de SQL del lote o el valor `null`
- Un puntero que apunta a un objeto SQLException o el valor `null`

Se emite una sola excepción BatchUpdateException para el lote completo. Como mínimo un objeto SQLException está encadenado al objeto BatchUpdateException. Los objetos SQLException están encadenados en el mismo orden que las sentencias correspondientes que se añadieron al lote. Para ayudarle a asociar los objetos SQLException con las sentencias del lote, el campo descriptivo del error de cada objeto SQLException comienza con este texto:

```
Error para elemento del lote #n:
```

n es el número de la sentencia dentro del lote.

Los avisos de SQL producidos durante la ejecución del lote no emiten excepciones BatchUpdateException. Para obtener información sobre avisos, utilice el método Statement.getWarnings para el objeto en el que ejecutó el método executeBatch. Luego puede obtener una descripción de error, el estado de SQL y el código de error correspondientes a cada objeto SQLWarning.

aviso de SQL

IBM Data Server Driver para JDBC y SQLJ acumula avisos cuando las sentencias de SQL devuelven códigos de SQL positivos, y cuando devuelven códigos de SQL iguales a 0 junto con estados de SQL distintos de cero.

La invocación de getWarnings hace que se recupere un objeto SQLWarning.

Importante: Cuando una llamada a Statement.executeUpdate o PreparedStatement.executeUpdate no repercute sobre las filas, IBM Data Server Driver para JDBC y SQLJ genera un SQLWarning con el código de error +100.

Cuando una llamada a ResultSet.next no devuelve ninguna fila, IBM Data Server Driver para JDBC y SQLJ no genera un SQLWarning.

Un objeto SQLWarning genérico contiene la información siguiente:
- Un objeto String que contiene una descripción del aviso o el valor nulo
- Un objeto String que contiene el SQLSTATE o el valor nulo
- Un valor `int` que contiene un código de error
- Un puntero que indica la ubicación del objeto SQLWarning siguiente o un valor nulo

Cuando se utiliza IBM Data Server Driver para JDBC y SQLJ, al igual que ocurre con un objeto SQLException, un objeto SQLWarning puede contener también información específica de DB2. La información específica de DB2 correspondiente a un objeto SQLWarning es la misma que la información específica de DB2 correspondiente a un objeto SQLException.

Manejo de una excepción de SQL cuando se utiliza IBM Data Server Driver para JDBC y SQLJ

Al igual que ocurre en todos los programas Java, el manejo de errores para aplicaciones JDBC se realiza utilizando bloques `try/catch`. Los métodos emiten excepciones cuando se producen errores, y el código contenido en el bloque `catch` maneja esas excepciones.

Acerca de esta tarea

Estos son los pasos básicos para el manejo de una excepción SQLException en un programa JDBC que se ejecuta con IBM Data Server Driver para JDBC y SQLJ:

Procedimiento

1. Proporcione al programa acceso a la interfaz com.ibm.db2.jcc.DB2Diagnosable y a la clase com.ibm.db2.jcc.DB2Sqlca. Puede calificar al completo todas las referencias a esos elementos o puede importarlos:
   ```
   import com.ibm.db2.jcc.DB2Diagnosable;
   import com.ibm.db2.jcc.DB2Sqlca;
   ```
2. Opcional: Durante una conexión con un servidor de datos, establezca la propiedad retrieveMessagesFromServerOnGetMessage en `true` si desea recibir un texto de mensaje completo al invocar SQLException.getMessage.
3. Opcional: Durante una conexión IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 con una fuente de datos DB2 para z/OS, establezca la propiedad extendedDiagnosticLevel en `EXTENDED_DIAG_MESSAGE_TEXT` (241) si desea recibir información de diagnóstico ampliada similar a la información proporcionada por la sentencia de SQL GET DIAGNOSTICS al invocar SQLException.getMessage.
4. Coloque código que pueda generar una excepción SQLException en un bloque `try`.
5. En el bloque `catch`, ejecute los pasos siguientes en un bucle:
   a. Determine si ha recuperado la última excepción SQLException. En caso negativo, continúe en el paso siguiente.
   b. Opcional: Para una sentencia de SQL que se ejecuta en una fuente de datos IBM Informix, ejecute el método com.ibm.db2.jcc.DB2Statement.getIDSSQLStatementOffSet para determinar las columnas con errores de sintaxis.

DB2Statement.getIDSSQLStatementOffSet devuelve el desplazamiento dentro de la sentencia de SQL donde está situado el primer error de sintaxis.

c. Opcional: Para una sentencia de SQL que se ejecute en una fuente de datos de IBM Informix, ejecute el método SQLException.getCause para recuperar todos los mensajes de error de ISAM.
   1) Si la instancia Throwable que devuelve SQLException.getCause no es nula, realice uno de los siguientes conjuntos de pasos:
      - Emita SQLException.printStackTrace para imprimir un mensaje de error que incluya el texto del mensaje de error de ISAM. El texto del mensaje de error de ISAM va precedido de la serie "Caused by:" (causado por).
      - Recupere el código de error y el texto de mensaje para el mensaje de ISAM:
        a) Compruebe si Throwable es una instancia de una SQLException. Si es así, recupere el código de error de SQL de dicha SQLException.
        b) Ejecute el método Throwable.getMessage para recuperar el texto del mensaje de ISAM.

d. Compruebe si existe información específica de IBM Data Server Driver para JDBC y SQLJ comprobando si SQLException es una instancia de DB2Diagnosable. En caso afirmativo:
   1) Convierta el objeto en un objeto DB2Diagnosable.
   2) Opcional: invoque el método DB2Diagnosable.printTrace para escribir toda la información sobre SQLException en un objeto java.io.PrintWriter.
   3) Invoque el método DB2Diagnosable.getThrowable para determinar si un objeto java.lang.Throwable asociado ha causado la excepción SQLException.
   4) Invoque el método DB2Diagnosable.getSqlca para recuperar el objeto DB2Sqlca.
   5) Invoque el método DB2Sqlca.getSqlCode para recuperar un valor de código de error de SQL.
   6) Invoque el método DB2Sqlca.getSqlErrmc para recuperar una serie de caracteres que contiene todos los valores SQLERRMC, o invoque el método DB2Sqlca.getSqlErrmcTokens para recuperar los valores SQLERRMC dentro de una matriz.
   7) Invoque el método DB2Sqlca.getSqlErrp para recuperar el valor SQLERRP.
   8) Invoque el método DB2Sqlca.getSqlErrd para recuperar los valores SQLERRD dentro de una matriz.
   9) Invoque el método DB2Sqlca.getSqlWarn para recuperar los valores SQLWARN dentro de una matriz.
   10) Invoque el método DB2Sqlca.getSqlState para recuperar el valor SQLSTATE.
   11) Invoque el método DB2Sqlca.getMessage para recuperar el texto del mensaje de error procedente de la fuente de datos.

e. Invoque el método SQLException.getNextException para recuperar la excepción SQLException siguiente.

Ejemplo

El código de programa siguiente muestra cómo obtener información específica de IBM Data Server Driver para JDBC y SQLJ a partir de un SQLException proporcionado con IBM Data Server Driver para JDBC y SQLJ. Los números situados a la derecha de determinadas sentencias corresponden a pasos descritos anteriormente.

Figura 20. Proceso de una excepción de SQL cuando se utiliza IBM Data Server Driver para JDBC y SQLJ

```
import java.sql.*;           // Importar paquete de la API de JDBC
import com.ibm.db2.jcc.DB2Diagnosable;  // Importar paquetes para DB2    1
import com.ibm.db2.jcc.DB2Sqlca;        // Soporte de SQLException
java.io.PrintWriter printWriter;        // Para volcar toda la información
                                        // de excepciones de SQL
String url = "jdbc:db2://myhost:9999/myDB:" +                           2
  "retrieveMessagesFromServerOnGetMessage=true;";
                                  // Establecer propiedades para recuperar todo el
                                  // texto del mensaje
String user = "db2adm";
String password = "db2adm";
java.sql.Connection con =
  java.sql.DriverManager.getConnection (url, user, password)
                                  // Conectar con una fuente de datos DB2 para z/OS


...
try {                                                                   4
  // Código que podría generar excepciones de SQL
 ...
} catch(SQLException sqle) {
    while(sqle != null) {             // Comprobar si hay más    5a
                                      // excepciones de SQL para procesar
 //=====> Proceso de errores opcional sólo de IBM Data Server Driver para
 // JDBC y SQLJ
      if (sqle instanceof DB2Diagnosable) {                             5d
      // Comprobar si existe información específica de
      //  IBM Data Server Driver para JDBC y SQLJ
          com.ibm.db2.jcc.DB2Diagnosable diagnosable =
            (com.ibm.db2.jcc.DB2Diagnosable)sqle;                       5d1
          diagnosable.printTrace (printWriter, "");                     5d2
          java.lang.Throwable throwable =
            diagnosable.getThrowable();                                 5d3
          if (throwable != null) {
            // Extraer información sobre java.lang.Throwable,
            // tal como mensaje o rastreo de pila.
            ...
          }
          DB2Sqlca sqlca = diagnosable.getSqlca();                      5d4
                                        // Obtener objeto DB2Sqlca
          if (sqlca != null) {            // Comprobar que DB2Sqlca no es nulo
            int sqlCode = sqlca.getSqlCode(); // Obtener el código de error de
                                              // SQL  5d5
            String sqlErrmc = sqlca.getSqlErrmc();                      5d6
                                        // Obtener el valor SQLERRMC completo
            String[] sqlErrmcTokens = sqlca.getSqlErrmcTokens();
                                        // También se pueden recuperar
                                        // distintivos SQLERRMC individuales
            String sqlErrp = sqlca.getSqlErrp();                        5d7
                                        // Obtener el valor SQLERRP
            int[] sqlErrd = sqlca.getSqlErrd();                         5d8
                                        // Obtener campos SQLERRD
            char[] sqlWarn = sqlca.getSqlWarn();                        5d9
                                        // Obtener campos SQLWARN
```

```
            String sqlState = sqlca.getSqlState();                    5d10
                                        // Obtener SQLSTATE
            String errMessage = sqlca.getMessage();                   5d11
                                        // Obtener mensaje de error

            System.err.println ("Mensaje de error de servidor: " + errMessage);

            System.err.println ("--------------- SQLCA ---------------");
            System.err.println ("Código de error: " + sqlCode);
            System.err.println ("SQLERRMC: " + sqlErrmc);
            If (sqlErrmcTokens != null) {
              for (int i=0; i< sqlErrmcTokens.length; i++) {
                System.err.println ("  token " + i + ": " + sqlErrmcTokens[i]);
              }
            }
            System.err.println ( "SQLERRP: " + sqlErrp );
            System.err.println (
              "SQLERRD(1): " + sqlErrd[0] + "\n" +
              "SQLERRD(2): " + sqlErrd[1] + "\n" +
              "SQLERRD(3): " + sqlErrd[2] + "\n" +
              "SQLERRD(4): " + sqlErrd[3] + "\n" +
              "SQLERRD(5): " + sqlErrd[4] + "\n" +
              "SQLERRD(6): " + sqlErrd[5] );
            System.err.println (
              "SQLWARN1: " + sqlWarn[0] + "\n" +
              "SQLWARN2: " + sqlWarn[1] + "\n" +
              "SQLWARN3: " + sqlWarn[2] + "\n" +
              "SQLWARN4: " + sqlWarn[3] + "\n" +
              "SQLWARN5: " + sqlWarn[4] + "\n" +
              "SQLWARN6: " + sqlWarn[5] + "\n" +
              "SQLWARN7: " + sqlWarn[6] + "\n" +
              "SQLWARN8: " + sqlWarn[7] + "\n" +
              "SQLWARN9: " + sqlWarn[8] + "\n" +
              "SQLWARNA: " + sqlWarn[9] );
            System.err.println ("SQLSTATE: " + sqlState);
                                        // Porción de excepción de SQL
          }
        sqle=sqle.getNextException();     // Recuperar excepción
                                          // de SQL siguiente       5e
      }
}
```

Manejo de un aviso de SQL cuando se utiliza IBM Data Server Driver para JDBC y SQLJ

A diferencia de los errores de SQL, los avisos de SQL no hacen que los métodos de JDBC emitan excepciones. En lugar de ello, las clases Connection, Statement, PreparedStatement, `CallableStatement` y ResultSet contienen métodos getWarnings, que es necesario invocar después de ejecutar sentencias de SQL para determinar si se han producido avisos de SQL.

Acerca de esta tarea

Estos son los pasos básicos para recuperar información de aviso de SQL:

Procedimiento

1. Opcional: Durante la conexión con el servidor de bases de datos, defina las propiedades que afectan a los objetos SQLWarning.

   Si desea recibir un texto de mensaje completo de un servidor de datos al ejecutar llamadas SQLWarning.getMessage, establezca la propiedad retrieveMessagesFromServerOnGetMessage en `true`.

Si está utilizando IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 con una fuente de datos DB2 para z/OS y desea obtener información de diagnóstico ampliada similar a la información proporcionada por la sentencia de SQL GET DIAGNOSTICS al realizar llamadas SQLWarning.getMessage, establezca la propiedad extendedDiagnosticLevel en EXTENDED_DIAG_MESSAGE_TEXT (241).

2. Inmediatamente después de invocar un método para conectar con un servidor de bases de datos o ejecutar una sentencia de SQL, invoque el método getWarnings para recuperar un objeto SQLWarning.
3. Ejecute en bucle los pasos siguientes:
   a. Determine si el objeto SQLWarning es nulo. En caso negativo, continúe en el paso siguiente.
   b. Invoque el método SQLWarning.getMessage para obtener la descripción del aviso.
   c. Invoque el método SQLWarning.getSQLState para obtener el valor SQLSTATE.
   d. Invoque el método SQLWarning.getErrorCode para obtener el valor del código de error.
   e. Si desea recibir información de aviso específica de DB2, siga los mismos pasos que realiza para obtener información específica de DB2 para un SQLException.
   f. Invoque el método SQLWarning.getNextWarning para recuperar el objeto SQLWarning siguiente.

Ejemplo

El código siguiente muestra cómo obtener información genérica sobre SQLWarning. Los números situados a la derecha de determinadas sentencias corresponden a pasos descritos anteriormente.

```
String url = "jdbc:db2://myhost:9999/myDB:" +                          1
  "retrieveMessagesFromServerOnGetMessage=true;";
                                  // Establecer propiedades para recuperar todo el
                                  // texto del mensaje
String user = "db2adm";
String password = "db2adm";
java.sql.Connection con =
  java.sql.DriverManager.getConnection (url, user, password)
                                  // Conectar con una fuente de datos DB2 para z/OS
Statement stmt;
ResultSet rs;
SQLWarning sqlwarn;
...
stmt = con.createStatement();         // Crear un objeto Statement
rs = stmt.executeQuery("SELECT * FROM EMPLOYEE");
                                  // Obtener tabla de resultados de la consulta
sqlwarn = stmt.getWarnings();     // Obtener avisos producidos             2
while (sqlwarn != null) {         // Mientras haya avisos, obtener e      3a
                                  // imprimir información de aviso
  System.out.println ("Descripción del aviso: " + sqlwarn.getMessage());  3b
  System.out.println ("SQLSTATE: " + sqlwarn.getSQLState());              3c
  System.out.println ("Código de error: " + sqlwarn.getErrorCode());      3d
  sqlwarn=sqlwarn.getNextWarning();     // Obtener aviso de SQL siguiente 3f
}
```

Figura 21. Ejemplo de proceso de un aviso de SQL

Recuperación de información de una excepción BatchUpdateException

Cuando se produce un error durante la ejecución de una sentencia de un lote de sentencias, el proceso continúa. Pero executeBatch emite una excepción BatchUpdateException.

Acerca de esta tarea

Para recuperar información de la excepción BatchUpdateException, siga estos pasos:

Procedimiento

1. Utilice el método BatchUpdateException.getUpdateCounts para determinar el número de filas que cada sentencia de SQL del lote ha actualizado antes de que se produjera la excepción.

   getUpdateCount devuelve una matriz con un elemento por cada sentencia del lote. Un elemento tiene uno de los valores siguientes:

   n El número de filas que la sentencia ha actualizado.

   Statement.SUCCESS_NO_INFO
   Este valor se devuelve si no se puede determinar el número de filas actualizado. No se puede determinar el número de filas si se cumplen las condiciones siguientes:
   - La aplicación está conectada con un subsistema que está en DB2 para z/OS Versión 8 en modalidad de función nueva, o posteriores.
   - La aplicación utiliza la Versión 3.1 o posteriores de IBM Data Server Driver para JDBC y SQLJ.
   - IBM Data Server Driver para JDBC y SQLJ utiliza operaciones INSERT de varias filas para ejecutar las actualizaciones por lotes.

   Statement.EXECUTE_FAILED
   Este valor se devuelve si la sentencia no se ha ejecutado satisfactoriamente.
2. Si la sentencia de proceso por lotes puede devolver claves generadas automáticamente:
   a. Convierta BatchUpdateException a com.ibm.db2.jcc.DBBatchUpdateException.
   b. Invoque el método DBBatchUpdateException.getDBGeneratedKeys para recuperar una matriz de objetos ResultSet que contiene las claves generadas automáticamente para cada ejecución de la sentencia de SQL de proceso por lotes.
   c. Pruebe si cada objeto ResultSet de la matriz es nulo.

      Cada ResultSet contiene:
      - Si el ResultSet no es nulo, contiene las claves generadas automáticamente correspondientes a una ejecución de la sentencia de SQL de proceso por lotes.
      - Si ResultSet es nulo, ha fallado la ejecución de la sentencia de proceso por lotes.
3. Utilice los métodos getMessage, getSQLState y getErrorCode de SQLException para obtener la descripción del error, el estado de SQL y el código de error correspondientes al primer error.

4. Utilice el método BatchUpdateException.getNextException para obtener una excepción SQLException encadenada.
5. Ejecute en bucle las llamadas de método getMessage, getSQLState, getErrorCode y getNextException para obtener información sobre una excepción SQLException y obtener la siguiente excepción SQLException.

Ejemplo

El siguiente fragmento de código de programa muestra cómo obtener los campos de una excepción BatchUpdateException y los objetos encadenados SQLException para una sentencia de proceso por lotes que devuelve claves generadas automáticamente. El ejemplo presupone que sólo hay una columna en la clave generada automáticamente, y siempre hay exactamente un valor de clave, cuyo tipo de datos es numérico. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
try {
  // Actualizaciones por lotes
} catch(BatchUpdateException buex) {
    System.err.println("Contents of BatchUpdateException:");
    System.err.println(" Update counts: ");
    int [] updateCounts = buex.getUpdateCounts();             1
    for (int i = 0; i < updateCounts.length; i++) {
      System.err.println("  Statement " + i + ":" + updateCounts[i]);
    }
    ResultSet[] resultList =
      ((DBBatchUpdateException)buex).getDBGeneratedKeys();    2
    for (i = 0; i < resultList.length; i++) {
      if (resultList[i] == null)
         continue; // Pasar por alto el ResultSet para el que se produjo un error
      else {
        rs.next();
        java.math.BigDecimal idColVar = rs.getBigDecimal(1);
                                      // Obtener valor de clave generada
                                      // automáticamente
        System.out.println("Valor de clave generada automáticamente = " + idColVar);
      }
    }
    System.err.println(" Message: " + buex.getMessage());     3
    System.err.println(" SQLSTATE: " + buex.getSQLState());
    System.err.println(" Error code: " + buex.getErrorCode());
    SQLException ex = buex.getNextException();                4
    while (ex != null) {                                      5
      System.err.println("SQL exception:");
      System.err.println(" Message: " + ex.getMessage());
      System.err.println(" SQLSTATE: " + ex.getSQLState());
      System.err.println(" Error code: " + ex.getErrorCode());
      ex = ex.getNextException();
    }
}
```

Desconexión respecto de fuentes de datos en aplicaciones JDBC

Una vez finalizada una conexión con una fuente de datos, es *esencial* que cierre la conexión con la fuente de datos. De esta manera se libera inmediatamente la base de datos y los recursos JDBC del objeto Connection.

Acerca de esta tarea

Para cerrar la conexión con la fuente de datos, utilice el método close. Por ejemplo:

```
 Connection con;
 ...
con.close();
```

Para una conexión con una fuente de datos DB2, si la modalidad de confirmación automática no está activa, es necesario que la conexión se encuentre en un límite de unidad de trabajo para poder cerrar la conexión.

Para una conexión con una base de datos IBM Informix, si la base de datos es compatible con la anotación cronológica y la modalidad de confirmación automática no está activa, es necesario que la conexión se encuentre en un límite de unidad de trabajo para poder cerrar la conexión.

Capítulo 4. Programación de aplicaciones SQLJ

La escritura de una aplicación SQLJ tiene mucho en común con la escritura de una aplicación SQL en cualquier otro lenguaje.

En general, es necesario que realice las acciones siguientes:
- Importe los paquetes de Java donde residen los métodos de SQLJ y JDBC.
- Declare variables para enviar datos a tablas de DB2 o recuperar datos de ellas.
- Conecte con una fuente de datos.
- Ejecute sentencias de SQL.
- Trate los errores y avisos de SQL.
- Desconecte de la fuente de datos.

Aunque las tareas que necesita realizar son similares a las que se ejecutan para otros lenguajes, la forma de ejecutarlas y el orden de ejecución es algo diferente.

Ejemplo de una aplicación SQLJ simple

Aplicación SQLJ simple que muestra los elementos básicos que es necesario incluir en una aplicación JDBC.

Figura 22. Aplicación SQLJ sencilla

```
import sqlj.runtime.*;                                          1
import java.sql.*;

#sql context EzSqljCtx;                                         3a
#sql iterator EzSqljNameIter (String LASTNAME);                 4a

public class EzSqlj {
  public static void main(String args[])
    throws SQLException
  {
    EzSqljCtx ctx = null;
    String URLprefix = "jdbc:db2:";
    String url;
    url = new String(URLprefix + args[0]);
                                          // El nombre de ubicación es un
                                          // parámetro de entrada
    String hvmgr="000010";                                      2
    String hvdeptno="A00";
    try {                                                       3b
      Class.forName("com.ibm.db2.jcc.DB2Driver");
    } catch (Exception e)
    {
       throw new SQLException("Error en EzSqlj: no se pudo cargar controlador");
    }
    try
    {
       System.out.println("Se va a conectar utilizando el url: " + url);
       Connection con0 = DriverManager.getConnection(url);      3c
                                          // Crear una conexión JDBC
       con0.setAutoCommit(false);         // Desactivar la confirmación automática
       ctx = new EzSqljCtx(con0);                               3d

       try
       {
```

```
        EzSqljNameIter iter;
        int count=0;

        #sql [ctx] iter =
          {SELECT LASTNAME FROM EMPLOYEE};                          4b
                                          // Crear tabla de resultados de SELECT
        while (iter.next()) {                                       4c
           System.out.println(iter.LASTNAME()); // Obtener filas tabla resultados
           count++;
        }
        System.out.println("Recuperadas " + count + " filas de datos");
        iter.close();        // Cerrar el iterador
      }
      catch( SQLException e )                                       5
      {
         System.out.println ("**** SELECT SQLException...");
         while(e!=null) {
           System.out.println ("Mensaje de error: " + e.getMessage());
           System.out.println ("SQLSTATE: " + e.getSQLState());
           System.out.println ("Código de error: " + e.getErrorCode());
           e = e.getNextException(); // Buscar excepciones encadenadas
         }
      }
      catch (Exception e)
      {
         System.out.println("**** Excepción no de SQL  = " + e);
         e.printStackTrace();
      }
      try
      {
        #sql [ctx]                                                  4d
          {UPDATE DEPARTMENT SET MGRNO=:hvmgr
             WHERE DEPTNO=:hvdeptno};   // Actualizar datos para un departamento
                                                                    6
        #sql [ctx] {COMMIT}; // Confirmar actualización
      }
      catch (SQLException e)
      {
         System.out.println ("**** UPDATE SQLException...");
         System.out.println ("Error msg: " + e.getMessage() + ". SQLSTATE=" +
           e.getSQLState() + "Código de error=" + e.getErrorCode());
         e.printStackTrace();
      }
      catch (Exception e)
      {
         System.out.println("**** Excepción no de SQL  = " + e);
         e.printStackTrace();
      }
      ctx.close();                                                  7
    }
    catch (SQLException e)
    {
       System.out.println ("**** SQLException ...");
       System.out.println ("Error msg: " + e.getMessage() + ". SQLSTATE=" +
         e.getSQLState() + "Código de error=" + e.getErrorCode());
       e.printStackTrace();
    }
    catch (Exception e)
    {
      System.out.println("**** Excepción no de SQL  = " + e);
      e.printStackTrace();
    }

}
```

Notas para la Figura 22 en la página 143:

Nota	Descripción
1	Estas sentencias importan el paquete java.sql, que contiene la API básica de JDBC, y el paquete sqlj.runtime, que contiene la API de SQLJ. Para obtener información sobre otros paquetes o clases que puede ser necesario acceder, consulte "Paquetes Java para soporte de SQLJ".
2	Las variables hvmgr y hvdeptno de tipo `String` son *identificadores de lenguaje principal*, que son equivalentes a variables de lenguaje principal de DB2. Consulte "Variables en aplicaciones SQLJ" para obtener más información.
3a, 3b, 3c y 3d	Estas sentencias muestran cómo conectar con una fuente de datos utilizando una de las tres técnicas disponibles. Consulte "Conexión con una fuente de datos utilizando SQLJ" para conocer más detalles. El paso 3b (carga del controlador JDBC) no es necesario si se utiliza JDBC 4.0 o posterior.
4a, 4b, 4c y 4d	Estas sentencias muestran cómo ejecutar sentencias de SQL en SQLJ. La sentencia 4a es el equivalente de SQLJ para declarar un cursor de SQL. Las sentencias 4b y 4c son un equivalente de SQLJ para ejecutar sentencias FETCH de SQL y OPEN CURSOR de SQL. La sentencia 4d es el equivalente de SQLJ para ejecutar una sentencia UPDATE de SQL. Para obtener más información, consulte "Sentencias de SQL en una aplicación SQLJ".
5	Este bloque `try/catch` muestra el uso de la clase SQLException para el manejo de errores de SQL. Para obtener más información sobre el manejo de errores de SQL, consulte "Manejo de errores de SQL en una aplicación SQLJ". Para obtener más información sobre el manejo de avisos de SQL, consulte "Manejo de avisos de SQL en una aplicación SQLJ".
6	Esto es un ejemplo de un comentario. Para conocer las reglas por las que se rige la inclusión de comentarios en programas SQLJ, consulte "Comentarios en una aplicación SQLJ".
7	Esta sentencia cierra la conexión con la fuente de datos. Consulte "Cierre de la conexión con la fuente de datos en una aplicación SQLJ".

Conexión a una fuente de datos utilizando SQLJ

En una aplicación SQLJ, al igual que en cualquier otra aplicación DB2, debe estar conectado a una fuente de datos para poder ejecutar sentencias de SQL.

Acerca de esta tarea

Dispone de seis técnicas para conectar con una fuente de datos en un programa SQLJ. Dos de estas técnicas utilizan la interfaz DriverManager de JDBC, otras dos utilizan la interfaz DataSource de JDBC, una de las técnicas utiliza un contexto de conexión creado previamente, y otra técnica utiliza la conexión por omisión.

Técnica de conexión 1 de SQLJ: interfaz DriverManager de JDBC

La técnica de conexión 1 de SQLJ utiliza la interfaz DriverManager de JDBC como medio subyacente para crear la conexión.

Acerca de esta tarea

Para utilizar la técnica de conexión 1 de SQLJ, siga estos pasos:

Procedimiento

1. Ejecute una *cláusula de declaración de conexión* de SQLJ.

Esto genera una *clase de contexto de conexión*. El formato más sencillo de la cláusula de declaración de conexión es el siguiente:

```
#sql context nombre_clase_contexto;
```

El nombre de la clase de contexto de conexión que se genera es *nombre_clase_contexto*.

2. Cargue un controlador JDBC invocando el método Class.forName.
   - Para IBM Data Server Driver para JDBC y SQLJ, invoque Class.forName del modo siguiente:

     ```
     Class.forName("com.ibm.db2.jcc.DB2Driver");
     ```

     Este paso no es necesario si utiliza el controlador JDBC 4.0 o posterior.
3. Invoque el constructor para la clase de contexto de conexión que creó en el paso 1 en la página 145.

   Esto crea un objeto de contexto de conexión que especificará en cada sentencia de SQL que ejecute en la fuente de datos asociada. La sentencia de invocación del constructor debe tener uno de los formatos siguientes:

   ```
   clase-contexto-conexión objeto-contexto-conexión=
     new clase_contexto_conexión
   (String url, boolean confirmación_automática);

   clase-contexto-conexión objeto-contexto-conexión=
     new clase-contexto-conexión(String  url, String usuario,
       String contraseña, boolean confirmación-automática);
   clase-contexto-conexión objeto-contexto-conexión=
     new clase-contexto-conexión(String url, Properties info,
       boolean confirmación-automática);
   ```

   El significado de los parámetros es el siguiente:

   url
   : Es una serie de caracteres que especifica el nombre de ubicación correspondiente a la fuente de datos. El formato de este argumento es uno de los especificados en "Conexión con una fuente de datos utilizando la interfaz DriverManager con IBM Data Server Driver para JDBC y SQLJ". El formato depende del controlador JDBC que esté utilizando.

   usuario **y** *contraseña*
   : Especifique un ID de usuario y una contraseña para conectar con la fuente de datos, si ésta los necesita.

   info
   : Especifica un objeto de tipo java.util.Properties que contiene un conjunto de propiedades de controlador correspondientes a la conexión. Para IBM Data Server Driver para JDBC y SQLJ, puede especificar cualquiera de las propiedades listadas en "Propiedades para el IBM Data Server Driver para JDBC y SQLJ".

   confirmación-automática
   : Especifica si el gestor de bases de datos debe emitir una operación de confirmación después de cada sentencia. Los valores posibles son `true` o `false`. Si especifica `false`, necesitará realizar operaciones de confirmación explícitas.

Ejemplo

El código siguiente utiliza la técnica de conexión 1 para crear una conexión con la ubicación NEWYORK. La conexión exige especificar un ID de usuario y contraseña, y no necesita confirmación automática. Los números que aparecen a la

derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
#sql context Ctx;      // Crear clase de contexto de conexión Ctx      1
String userid="dbadm";         // Declarar variables para ID de usuario y contraseña
String password="dbadm";
String empname;             // Declarar una variable de lenguaje principal
...
try {                      // Cargar el controlador JDBC
  Class.forName("com.ibm.db2.jcc.DB2Driver");                          2
}
catch (ClassNotFoundException e) {
   e.printStackTrace();
}
Ctx myConnCtx=                                                         3
  new Ctx("jdbc:db2://sysmvs1.stl.ibm.com:5021/NEWYORK",
  userid,password,false);  // Crear objeto contexto conexión myConnCtx
                            // para la conexión con NEWYORK
#sql [myConnCtx] {SELECT LASTNAME INTO :empname FROM EMPLOYEE
   WHERE EMPNO='000010'};
                            // Usar myConnCtx para ejec. sentencia de SQL
```

Figura 23. Uso de la técnica de conexión 1 para conectar con una fuente de datos

Técnica de conexión 2 de SQLJ: interfaz DriverManager de JDBC

La técnica de conexión 2 de SQLJ utiliza la interfaz DriverManager de JDBC como medio subyacente para crear la conexión.

Acerca de esta tarea

Para utilizar la técnica de conexión 2 de SQLJ, siga estos pasos:

Procedimiento

1. Ejecute una *cláusula de declaración de conexión* de SQLJ.

   Esto genera una *clase de contexto de conexión*. El formato más sencillo de la cláusula de declaración de conexión es el siguiente:

   ```
   #sql context nombre_clase_contexto;
   ```

   El nombre de la clase de contexto de conexión que se genera es *nombre_clase_contexto*.

2. Cargue un controlador JDBC invocando el método Class.forName.
   - Para IBM Data Server Driver para JDBC y SQLJ, invoque Class.forName del modo siguiente:

     ```
     Class.forName("com.ibm.db2.jcc.DB2Driver");
     ```

     Este paso no es necesario si utiliza el controlador JDBC 4.0 o posterior.

3. Invoque el método DriverManager.getConnection de JDBC.

   Esto crea un objeto de conexión de JDBC para la conexión con la fuente de datos. Puede utilizar cualquiera de las modalidades de getConnection que están especificadas en "Conexión con una fuente de datos utilizando la interfaz DriverManager con IBM Data Server Driver para JDBC y SQLJ".

   Los significados de los parámetros *url*, *usuario* y *contraseña* son:

   url

   Es una serie de caracteres que especifica el nombre de ubicación correspondiente a la fuente de datos. El formato de este argumento es uno de los especificados en "Conexión con una fuente de datos utilizando la

interfaz DriverManager con IBM Data Server Driver para JDBC y SQLJ". El formato depende del controlador JDBC que esté utilizando.

usuario **y** *contraseña*
Especifique un ID de usuario y una contraseña para conectar con la fuente de datos, si ésta los necesita.

4. Invoque el constructor para la clase de contexto de conexión que creó en el paso 1 en la página 147

   Esto crea un objeto de contexto de conexión que especificará en cada sentencia de SQL que ejecute en la fuente de datos asociada. La sentencia de invocación del constructor debe tener el formato siguiente:

   ```
   clase-contexto-conexión objeto-contexto-conexión=
     new clase-contexto-conexión(Connection objeto-conexión-JDBC);
   ```

   El parámetro *objeto_conexión_JDBC* es el objeto Connection que creó en el paso 3 en la página 147.

Ejemplo

El código siguiente utiliza la técnica de conexión 2 para crear una conexión con la ubicación NEWYORK. La conexión exige especificar un ID de usuario y contraseña, y no necesita confirmación automática. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
#sql context Ctx;          // Crear clase de contexto de conexión Ctx   1
String userid="dbadm";         // Declarar variables para ID de usuario y contraseña
String password="dbadm";
String empname;             // Declarar una variable de lenguaje principal
...
try {                       // Cargar el controlador JDBC
  Class.forName("com.ibm.db2.jcc.DB2Driver");                          2
}
catch (ClassNotFoundException e) {
   e.printStackTrace();
}
Connection jdbccon=                                                    3
  DriverManager.getConnection("jdbc:db2://sysmvs1.stl.ibm.com:5021/NEWYORK",
    userid,password);
                            // Crear objeto de conexión jdbccon de JDBC
jdbccon.setAutoCommit(false); // No realizar confirmación automática
Ctx myConnCtx=new Ctx(jdbccon);                                        4
                            // Crear objeto de contexto de conexión myConnCtx
                            // para la conexión con NEWYORK
#sql [myConnCtx] {SELECT LASTNAME INTO :empname FROM EMPLOYEE
   WHERE EMPNO='000010'};
                            // Usar myConnCtx para ejec. sentencia de SQL
```

Figura 24. Uso de la técnica de conexión 2 para conectar con una fuente de datos

Técnica de conexión 3 de SQLJ: interfaz DataSource de JDBC

La técnica de conexión 3 de SQLJ utiliza la interfaz DataSource de JDBC como medio subyacente para crear la conexión.

Acerca de esta tarea

Para utilizar la técnica de conexión 3 de SQLJ, siga estos pasos:

Procedimiento

1. Ejecute una *cláusula de declaración de conexión* de SQLJ.

Esto genera una *clase de contexto de conexión*. El formato más sencillo de la cláusula de declaración de conexión es el siguiente:

```
#sql context nombre_clase_contexto;
```

El nombre de la clase de contexto de conexión que se genera es *nombre_clase_contexto*.

2. Si el administrador del sistema ha creado un objeto DataSource en un programa diferente, siga estos pasos. En otro caso, cree un objeto DataSource y asigne propiedades al objeto.
   a. Obtenga el nombre lógico de la fuente de datos con la que necesita conectar.
   b. Cree un contexto para utilizarlo en el paso siguiente.
   c. En el programa de aplicación, utilice Java Naming and Directory Interface (JNDI) para obtener el objeto DataSource asociado al nombre lógico de fuente de datos.
3. Invoque el método DataSource.getConnection de JDBC.

   Esto crea un objeto de conexión de JDBC para la conexión con la fuente de datos. Puede utilizar uno de los formatos siguientes de getConnection:

   ```
   getConnection();
   getConnection(usuario, contraseña);
   ```

   Los significados de los parámetros *usuario* y *contraseña* son:

   usuario **y** *contraseña*
   Especifique un ID de usuario y una contraseña para conectar con la fuente de datos, si ésta los necesita.
4. Si el valor por omisión para la confirmación automática no es apropiado, invoque el método Connection.setAutoCommit.

   De esta forma especifica si el gestor de bases de datos debe emitir una operación de confirmación después de cada sentencia. El formato de este método es:

   ```
   setAutoCommit(boolean autocommit);
   ```
5. Invoque el constructor para la clase de contexto de conexión que creó en el paso 1 en la página 148.

   Esto crea un objeto de contexto de conexión que especificará en cada sentencia de SQL que ejecute en la fuente de datos asociada. La sentencia de invocación del constructor debe tener el formato siguiente:

   ```
   clase-contexto-conexión objeto-contexto-conexión=
     new clase-contexto-conexión(Connection objeto-conexión-JDBC);
   ```

   El parámetro *objeto_conexión_JDBC* es el objeto Connection que creó en el paso 3.

Ejemplo

El código de programa siguiente utiliza la técnica de conexión 3 para crear una conexión con una ubicación cuyo nombre lógico es `jdbc/sampledb`. En este ejemplo se supone que el administrador del sistema ha creado y desplegado un objeto DataSource al que se puede acceder mediante la función lookup de JNDI. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
import java.sql.*;
import javax.naming.*;
import javax.sql.*;
...
#sql context CtxSqlj;   // Crear clase de contexto de conexión CtxSqlj   1
Context ctx=new InitialContext();                                        2b
DataSource ds=(DataSource)ctx.lookup("jdbc/sampledb");                   2c
Connection con=ds.getConnection();                                       3
String empname;             // Declarar una variable de lenguaje principal
...
con.setAutoCommit(false);   // No realizar confirmación automática       4
CtxSqlj myConnCtx=new CtxSqlj(con);                                      5
                            // Crear objeto de contexto de conexión myConnCtx
#sql [myConnCtx] {SELECT LASTNAME INTO :empname FROM EMPLOYEE
   WHERE EMPNO='000010'};
                            // Usar myConnCtx para ejec. sentencia de SQL
```

Figura 25. Uso de la técnica de conexión 3 para conectar con una fuente de datos

Técnica de conexión 4 de SQLJ: interfaz DataSource de JDBC

La técnica de conexión 4 de SQLJ utiliza la interfaz DataSource de JDBC como medio subyacente para crear la conexión. Esta técnica **exige** que DataSource esté registrado en JNDI.

Acerca de esta tarea

Para utilizar la técnica de conexión 4 de SQLJ, siga estos pasos:

Procedimiento

1. Consulte al administrador del sistema para obtener el nombre lógico de la fuente de datos con la que necesita conectar.
2. Ejecute una cláusula de declaración de conexión de SQLJ.

   Para este tipo de conexión, la cláusula de declaración de conexión debe tener este formato:

   ```
   #sql public static context nombre_clase_contexto
    with (dataSource="nombre-lógico");
   ```

   El contexto de conexión debe estar declarado como public y static. *nombre_lógico* es el nombre de fuente de datos que obtuvo en el paso 1.
3. Invoque el constructor para la clase de contexto de conexión que creó en el paso 2.

   Esto crea un objeto de contexto de conexión que especificará en cada sentencia de SQL que ejecute en la fuente de datos asociada. La sentencia de invocación del constructor debe tener uno de los formatos siguientes:

   ```
   clase-contexto-conexión objeto-contexto-conexión=
     new clase_contexto_conexión();

   clase-contexto-conexión objeto-contexto-conexión=
     new clase_contexto_conexión (String usuario,
       String contraseña);
   ```

   Los significados de los parámetros *usuario* y *contraseña* son:

   usuario **y** *contraseña*
   : Especifique un ID de usuario y una contraseña para conectar con la fuente de datos, si ésta los necesita.

Ejemplo

El código siguiente utiliza la técnica de conexión 4 para crear una conexión con una ubicación cuyo nombre lógico es jdbc/sampledb. La conexión exige utilizar un ID de usuario y contraseña.

```
#sql public static context Ctx
  with (dataSource="jdbc/sampledb");                                    2
                              // Crear la clase de contexto de conexión Ctx
String userid="dbadm";        // Declarar variables para ID de usuario y contraseña
String password="dbadm";

String empname;             // Declarar una variable de lenguaje principal
...
Ctx myConnCtx=new Ctx(userid, password);                                3
                              // Crear objeto de contexto de conexión myConnCtx
                              // para la conexión con jdbc/sampledb
#sql [myConnCtx] {SELECT LASTNAME INTO :empname FROM EMPLOYEE
   WHERE EMPNO='000010'};
                              // Usar myConnCtx para ejec. sentencia de SQL
```

Figura 26. Uso de la técnica de conexión 4 para conectar con una fuente de datos

Técnica de conexión 5 de SQLJ: utilización de un contexto de conexión creado previamente

La técnica de conexión 5 de SQLJ utiliza un contexto de conexión creado previamente para conectar con la fuente de datos.

Acerca de esta tarea

En general, un programa declara una clase de contexto de conexión, crea contextos de conexión y los pasa como parámetros a otros programas. Un programa que utiliza el contexto de conexión invoca un constructor con el objeto de contexto de conexión pasado como argumento.

Ejemplo

El programa CtxGen.sqlj declara el contexto de conexión Ctx y crea la instancia oldCtx:

```
#sql context Ctx;
...
// Crear objeto contexto conexión oldCtx
```

El programa test.sqlj recibe oldCtx como parámetro y utiliza oldCtx como argumento de su constructor de contexto de conexión:

```
void useContext(sqlj.runtime.ConnectionContext oldCtx)
                                    // oldCtx se creó en CtxGen.sqlj
{
  Ctx myConnCtx=
    new Ctx(oldCtx);        // Crear objeto contexto conexión myConnCtx
                                    // a partir de oldCtx
  #sql [myConnCtx] {SELECT LASTNAME INTO :empname FROM EMPLOYEE
     WHERE EMPNO='000010'};
                                    // Usar myConnCtx para ejec. sentencia de SQL
...
}
```

Técnica de conexión 6 de SQLJ: Utilización de la conexión por omisión

La técnica de conexión 6 de SQLJ utiliza la conexión por omisión para conectar con la fuente de datos. Se debe utilizar únicamente en situaciones en las que la hebra de la base de datos está controlada por otro gestor de recursos, tal como el entorno de procedimiento almacenado de Java.

Acerca de esta tarea

Utilice la conexión por omisión especificando sus sentencias de SQL sin un objeto de contexto de conexión. Cuando utiliza esta técnica, no es necesario cargar un controlador JDBC a menos que utilice explícitamente interfaces de JDBC en su programa.

El contexto de conexión por omisión puede ser:

- El contexto de conexión correspondiente a la fuente de datos que está vinculada al nombre lógico jdbc/defaultDataSource
- Un contexto de conexión creado explícitamente que se ha configurado como contexto de conexión por omisión mediante el método ConnectionContext.setDefaultContext. No es recomendable utilizar este método para crear un contexto de conexión por omisión.

Ejemplo

La siguiente cláusula de ejecución de SQLJ no tiene un contexto de conexión, por lo que utiliza el contexto de conexión por omisión.

```
#sql {SELECT LASTNAME INTO :empname FROM EMPLOYEE
   WHERE EMPNO='000010'};  // Utilizar conexión por omisión
                           // para ejecutar una sentencia de SQL
```

Paquetes Java para el soporte SQLJ

Para ejecutar sentencias de SQLJ o invocar métodos de JDBC en un programa de SQLJ, necesita poder acceder a todos o a partes de diversos paquetes Java que contengan soporte para dichas sentencias.

Puede hacerlo importando los paquetes o clases específicas, o bien utilizando nombres de clase totalmente calificados. Puede necesitar los paquetes o clases siguientes para su programa de SQLJ:

sqlj.runtime
Contiene la API de ejecución de SQLJ.

java.sql
Contiene la API básica de JDBC.

com.ibm.db2.jcc
Contiene la implementación de JDBC y SQLJ específica del controlador.

javax.naming
Contiene métodos para ejecutar una búsqueda de JNDI (Java Naming and Directory Interface).

javax.sql
Contiene métodos para crear objetos DataSource.

Variables en aplicaciones SQLJ

En los programas DB2 escritos en otros lenguajes, utiliza variables de lenguaje principal para pasar datos entre el programa de aplicación y DB2. En los programas SQLJ, puede utilizar variables de lenguaje principal o *expresiones de lenguaje principal*.

Una expresión de lenguaje principal comienza con un signo de dos puntos (:). El signo de dos puntos va seguido por un identificador opcional de la modalidad del parámetro (IN, OUT o INOUT), el cual va seguido por una cláusula de expresión entre paréntesis.

Las variables de lenguaje principal y expresiones de lenguaje principal distinguen entre mayúsculas y minúsculas.

Una expresión compleja es un elemento de matriz o expresión Java cuya evaluación da como resultado un valor individual. Las expresiones complejas contenidas en una cláusula de SQLJ deben estar delimitadas por paréntesis.

Los ejemplos siguientes muestran cómo utilizar expresiones de lenguaje principal.

Ejemplo: declaración de un identificador Java y su utilización en una sentencia SELECT:

En este ejemplo, la sentencia que comienza con `#sql` tiene la misma función que una sentencia SELECT en otros lenguajes de programación. Esta sentencia asigna el apellido del empleado cuyo número de empleado es 000010 al identificador `empname` de Java.

```
String empname;
...
#sql [ctxt]
  {SELECT LASTNAME INTO :empname FROM EMPLOYEE WHERE EMPNO='000010'};
```

Ejemplo: declaración de un identificador Java y su utilización en una llamada de procedimiento almacenado:

En este ejemplo, la sentencia que comienza con `#sql` tiene la misma función que una sentencia CALL de SQL en otros lenguajes de programación. Esta sentencia utiliza el identificador `empno` de Java como parámetro de entrada del procedimiento almacenado `A`. La palabra clave `IN`, que precede a `empno`, especifica que `empno` es un parámetro de entrada. Para un parámetro de una sentencia CALL, el valor por omisión es `IN`. El calificador explícito o por omisión que indica cómo se utiliza el parámetro (IN, OUT o INOUT) debe coincidir con el valor correspondiente contenido en la definición de parámetro que ha especificado en la sentencia CREATE PROCEDURE para el procedimiento almacenado.

```
String empno = "0000010";
...
#sql [ctxt] {CALL A (:IN empno)};
```

Ejemplo: uso de una expresión compleja como identificador de lenguaje principal:

Este ejemplo utiliza la expresión compleja (((int)yearsEmployed++/5)*500) como expresión de lenguaje principal.

```
#sql [ctxt] {UPDATE EMPLOYEE
        SET BONUS=:(((int)yearsEmployed++/5)*500) WHERE EMPNO=:empID};
```

SQLJ ejecuta las acciones siguientes cuando procesa una expresión de lenguaje principal compleja:

- Evalúa cada una de las expresiones de lenguaje principal, de izquierda a derecha, antes de asignar sus respectivos valores a la base de datos.
- Evalúa los efectos secundarios, tales como operaciones con operadores sufijos, de acuerdo con las normas normales de Java. Todas las expresiones de lenguaje principal se evalúan totalmente antes de pasar cualquiera de sus valores a DB2.
- Utiliza normas de Java para el redondeo y truncamiento de datos.

Por tanto, si el valor de yearsEmployed es 6 antes de ejecutar la sentencia UPDATE, el valor que la sentencia UPDATE asigna a la columna BONUS es ((int)6/5)*500, lo que equivale a 500. Después de asignar 500 a BONUS, el valor de yearsEmployed se incrementa.

Restricciones para nombres de variables: existen dos series de caracteres con significados especiales en los programas SQLJ. Tenga en cuenta las restricciones siguientes cuando utilice esas series de caracteres en sus programas SQLJ:

- La serie __sJT_ es un prefijo reservado que se utiliza para los nombres de variables generados por SQLJ. No comience los siguientes tipos de nombres con __sJT_:
  - Nombres de expresiones de lenguaje principal
  - Nombres de variables Java declarados en bloques que incluyan sentencias de SQL ejecutables
  - Nombres de parámetros para métodos que contienen sentencias ejecutables de SQL
  - Nombres de campos en clases que contienen sentencias ejecutables de SQL, o en clases con subclases o clases incluidas que contienen sentencias ejecutables de SQL
- La serie _SJ es un sufijo reservado que se utiliza para archivos de recursos y clases que han sido creados por SQLJ. Evite utilizar la serie _SJ en nombres de clases y nombres de archivos fuente de entrada.

Variables de indicador en aplicaciones SQLJ

En programas SQLJ, las variables de indicador se pueden utilizar para pasar el valor NULL a un servidor de datos o de un servidor de datos, pasar el valor por omisión de una columna al servidor de datos o indicar que un valor de la variable del lenguaje principal no está asignado.

Una variable o expresión de lenguaje principal puede ir seguida de una variable de indicador. Una variable de indicador empieza por un signo de dos puntos (:) y tiene un tipo de datos short. Para la entrada, una variable de indicador indica si la variable o expresión de lenguaje principal correspondiente tiene el valor por omisión, un valor que no sea nulo, el valor nulo o bien no está asignado. Una variable no asignada en una sentencia de SQL tiene el mismo resultado que si la variable y su columna de destino no aparecieran en la sentencia de SQL. Para la salida, una variable de indicador indica dónde tiene un valor no nulo o un valor nulo la variable o expresión de lenguaje principal correspondiente.

En programas SQLJ, las variables de indicador que indican un valor nulo realizan la misma función que la asignación de un valor Java nulo a una columna de tabla. Sin embargo, es necesario utilizar una variable de indicador para recuperar el valor nulo de SQL de una tabla a una variable de lenguaje principal.

Las variables de indicador se pueden usar para asignar el valor DEFAULT o el valor UNASSIGNED a columnas con el fin de simplificar la codificación en las aplicaciones. Por ejemplo, si una tabla tiene cuatro columnas y necesita actualizar cualquier combinación de estas columnas, sin el uso de variables de indicador por omisión o variables de indicador no asignadas, necesitará 15 sentencias UPDATE para llevar a cabo todas las combinaciones de actualizaciones. Con variables de indicador por omisión y variables de indicador no asignadas, se puede utilizar una sentencia UPDATE con las cuatro columnas en la sentencia SET para llevar a cabo todas las actualizaciones posibles. Las variables de indicador se utilizan para indicar qué columnas desea establecer en los valores por omisión y qué columnas no desea actualizar.

Para la entrada, SQLJ ofrece soporte al uso de variables de indicador para sentencias INSERT, UPDATE o MERGE.

Si personaliza la aplicación SQLJ, puede asignar uno de los valores siguientes a una variable de indicador en una aplicación SQLJ para especificar el tipo de la variable de lenguaje principal de entrada correspondiente.

Valor de indicador	Constante equivalente	Significado del valor
-1	sqlj.runtime.ExecutionContext.DBNull	Nulo
-2, -3, -4, -6		Nulo
-5	sqlj.runtime.ExecutionContext.DBDefault	Por omisión
-7	sqlj.runtime.ExecutionContext.DBUnassigned	Sin asignar
valor-short >=0	sqlj.runtime.ExecutionContext.DBNonNull	No nulo

Si no personaliza la aplicación, puede asignar uno de los valores siguientes a una variable de indicador para especificar el tipo de la variable de lenguaje principal de entrada correspondiente.

Valor de indicador	Constante equivalente	Significado del valor
-1	sqlj.runtime.ExecutionContext.DBNull	Nulo
-7 <= *valor-short* < -1		Nulo
0	sqlj.runtime.ExecutionContext.DBNonNull	No nulo
valor-short >0		No nulo

Para la salida, SQLJ ofrece soporte al uso de variables de indicador para las sentencias siguientes:

- Parámetros CALL con OUT o INOUT
- FETCH *iterador* INTO *variable-lenguajeprincipal*
- SELECT ... INTO *variable-lenguajeprincipal-1,...variable-lenguajeprincipal-n*

SQLJ asigna uno de los valores siguientes a una variable de indicador para especificar si se ha recuperado un valor nulo de SQL en la variable de lenguaje principal correspondiente.

Valor de indicador	Constante equivalente	Significado del valor
-1	sqlj.runtime.ExecutionContext.DBNull	El valor recuperado es SQL NULL
0		El valor recuperado no es SQL NULL

Las variables de indicador no se pueden utilizar para actualizar los conjuntos de resultados. Para asignar valores nulos o valores por omisión a conjuntos de resultados o para indicar que las columnas no están asignadas, invoque ResultSet.updateObject en los objetos ResultSet subyacentes de JDBC de los iteradores de SQLJ.

Los ejemplos siguientes muestran cómo utilizar variables de indicador.

Todos los ejemplos requieren que el servidor de datos admita los indicadores ampliados.

Ejemplo de uso de indicadores para asignar el valor por omisión a columnas durante una operación INSERT:

En este ejemplo, las columnas MGRNO y LOCATION deben establecerse en los valores por omisión. Para ello, el código efectúa estos pasos:

1. Asigna el valor ExecutionContext.DBNonNull a las variables de indicador (deptInd, dNameInd, rptDeptInd) para las variables de lenguaje de principal de entrada (dept, dName, rptDept) que envían valores que no son por omisión a las columnas de destino.
2. Asigna el valor ExecutionContext.DBDefault a las variables de indicador (mgrInd, locnInd) para las variables de lenguaje principal de entrada (mgr, locn) que envían valores por omisión a las columnas de destino.
3. Ejecuta una sentencia INSERT con los pares de variable de lenguaje principal y variable de indicador como entrada.

Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
import sqlj.runtime.*;
...
String dept = "F01";
String dName = "SHIPPING";
String rptDept = "A00";
String mgr, locn = null;
short deptInd, dNameInd, mgrInd, rptDeptInd, locnInd;
// Establecer variables de indicador para dept, dName y rptDept en valores no nulos
deptInd = dNameInd = rptDeptInd = ExecutionContext.DBNonNull;   1
mgrInd = ExecutionContext.DBDefault;                            2
locnInd = ExecutionContext.DBDefault;
#sql [ctxt]                                                     3
  {INSERT INTO DEPARTMENT
    (DEPTNO, DEPTNAME, MGRNO, ADMRDEPT, LOCATION)
    VALUES (:dept :deptInd, :dName :dNameInd,:mgr :mgrInd,
    :rptDept :rptDeptInd, :locn :locnInd)};
```

Ejemplo de uso de indicadores para asignar el valor por omisión para que deje los valores de columna sin asignar durante una operación UPDATE:

En este ejemplo, en las filas para el departamento F01, la columna MGRNO debe establecerse en su valor por omisión, el valor de la columna DEPTNAME debe cambiarse a RECEIVING y las columnas DEPTNO, DEPTNAME, ADMRDEPT y LOCATION deben mantenerse sin modificar. Para ello, el código efectúa estos pasos:

1. Asigna el valor nuevo de la columna DEPTNAME a la variable de lenguaje principal de entrada dName.

2. Asigna el valor ExecutionContext.DBDefault a la variable de indicador (mgrInd) para la variable de lenguaje principal de entrada (mgr) que envía el valor por omisión a la columna de destino.
3. Asigna el valor ExecutionContext.DBUnassigned a las variables de indicador (deptInd, dNameInd, rptDeptInd y locnInd) para las variables de lenguaje principal de entrada (dept, dName, rptDept y locn) que la operación UPDATE debe mantener sin modificar.
4. Ejecuta una sentencia UPDATE con los pares de variable de lenguaje principal y variable de indicador como entrada.

Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
import sqlj.runtime.*;
...
String dept = null;
String dName = "RECEIVING";                                          1
String rptDept = null;
String mgr, locn = null;
short deptInd, dNameInd, mgrInd, rptDeptInd, locnInd;
dNameInd = ExecutionContext.DBNonNull;
mgrInd = ExecutionContext.DBDefault;                                 2
deptInd = rptDeptInd = locnInd = ExecutionContext.DBUnassigned;      3
#sql [ctxt]                                                          4
  {UPDATE DEPARTMENT
    SET DEPTNO = :dept :deptInd,
        DEPTNAME = :dName :dNameInd,
        MGRNO = :mgr :mgrInd,
        ADMRDEPT = :rptDept :rptDeptInd,
        LOCATION = :locn :locnInd
    WHERE DEPTNO = "F01"
  };
```

Ejemplo de uso de indicadores para recuperar valores NULL de columnas:

En este ejemplo, la columna HIREDATE puede devolver el valor NULL. Para ello, el código efectúa estos pasos:

1. Define una variable de indicador para indicar cuándo se devuelve el valor NULL de HIREDATE.
2. Ejecuta sentencias FETCH con los pares de variable de lenguaje principal y variable de indicador como salida.
3. Comprueba la variable de indicador para determinar si se devuelve un valor NULL.

Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
import sqlj.runtime.*;
...
#sql iterator ByPos(String, Date);  // Declarar iterador de posición ByPos
{
  ...
  ByPos positer;                    // Declarar objeto de clase ByPos
  String name = null;        // Declarar variables de lenguaje principal
  Date hrdate = null;
  short indhrdate = null;           // Declarar variable de indicador      1
  #sql [ctxt] positer =
    {SELECT LASTNAME, HIREDATE FROM EMPLOYEE};
                                    // Asignar tabla de resultados de SELECT
                                    // al objeto iterador positer
  #sql {FETCH :positer INTO :name, :hrdate :indhrdate };               2
                                    // Recuperar primera fila
```

```
  while (!positer.endFetch())   // Determinar si FETCH ha devuelto una fila
  { if(indhrdate == ExecutionContext.DBNonNull {                        3
     System.out.println(name + " fue contratado en " +
      hrdate); }
    else {
     System.out.println(name + " no tiene fecha de contratación "); }
    #sql {FETCH :positer INTO :name, :hrdate };
                                  // Obtener la fila siguiente
  }
  positer.close();                // Cerrar el iterador                  5
}
```

Ejemplo de asignación de valores por omisión a columnas de conjuntos de resultados:

En este ejemplo, la columna HIREDATE de un conjunto de resultados debe establecerse en el valor por omisión. Para ello, el código efectúa estos pasos:

1. Recupera el ResultSet subyacente del iterador que mantiene los datos recuperados.
2. Ejecuta el método ResultSet.updateObject con la constante DB2PreparedStatement.DB_PARAMETER_DEFAULT para asignar el valor por omisión a la columna de conjunto de resultados.

Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
#sql public iterator sensitiveUpdateIter
 implements sqlj.runtime.Scrollable, sqlj.runtime.ForUpdate
 with (sensitivity=sqlj.runtime.ResultSetIterator.SENSITIVE,
 updateColumns="LASTNAME, HIREDATE") (String, Date);

  String name;                   // Declarar variables de lenguaje principal
  Date hrdate;

sensitiveUpdateIter iter = null;
#sql [ctx] iter = { SELECT LASTNAME, HIREDATE FROM EMPLOYEE};

iter.next();

java.sql.ResultSet rs =  iter.getResultSet();                       1
rs.updateString("LASTNAME", "FORREST");
rs.updateObject
 (2, com.ibm.db2.jcc.DB2PreparedStatement.DB_PARAMETER_DEFAULT););  2,3
rs.updateRow();
iter.close();
```

Comentarios en una aplicación SQLJ

Para documentar su programa SQLJ, es necesario que incluya comentarios. Puede utilizar comentarios Java fuera de las sentencias de SQLJ y comentarios de SQL o Java en sentencias de SQLJ.

Puede incluir comentarios Java fuera de las cláusulas SQLJ allí donde el lenguaje Java lo permita. Dentro de una cláusula SQLJ, puede utilizar comentarios en las ubicaciones siguientes:

- Dentro de una expresión de lenguaje principal (especificados entre /* y */ o precedidos por //).
- Dentro de una sentencia de SQL en una cláusula ejecutable, si el servidor de datos da soporte a un comentario dentro de la sentencia de SQL.
  - Para las conexiones con servidores de datos DB2 o servidores de datos Informix, los comentarios pueden ubicarse:

- En cualquier punto del texto de la sentencia de SQL, especificados entre pares /* y */. Los pares /* y */ pueden anidarse.
- Al final del texto de la sentencia de SQL, y precedidos por dos guiones (--).

– Para las conexiones con servidores de datos Informix únicamente, los comentarios pueden especificarse entre pares de llave izquierda ({) y llave derecha (}).

Ejecución de sentencias de SQL en aplicaciones SQLJ

En un programa SQL tradicional, puede ejecutar sentencias de SQL para crear tablas, actualizar datos de tablas, recuperar datos de tablas, invocar procedimientos almacenados, o confirmar o retrotraer transacciones. En un programa SQLJ, puede también ejecutar esas sentencias, dentro de *cláusulas ejecutables* de SQLJ.

Una cláusula ejecutable puede tener uno de los formatos generales siguientes:

```
#sql [contexto-conexión] {sentencia-sql};
#sql [contexto-conexión,contexto-ejecución] {sentencia-sql};
#sql [contexto-ejecución] {sentencia-sql};
```

La especificación contexto-ejecución
En una cláusula ejecutable, debe especificar **siempre** un contexto de conexión explícito, con una sola excepción: no es necesario especificar un contexto de conexión explícito para una sentencia FETCH. Sólo debe incluir un contexto de ejecución para casos específicos. Consulte "Control de la ejecución de sentencias de SQL en SQLJ" para obtener información sobre cuándo necesita un contexto de ejecución.

La especificación contexto-conexión
En una cláusula ejecutable, si no especifica explícitamente un contexto de conexión, la cláusula ejecutable utiliza el contexto de conexión por omisión.

Creación y modificación de objetos de base de datos en una aplicación SQLJ

Utilice cláusulas ejecutables de SQLJ para ejecutar sentencias de definición de datos (CREATE, ALTER, DROP, GRANT, REVOKE) o para ejecutar sentencias INSERT, UPDATE de búsqueda o posición o DELETE de búsqueda o posición.

Ejemplo

Las sentencias ejecutables siguientes muestran una operación INSERT, una operación UPDATE de búsqueda y una operación DELETE de búsqueda:

```
#sql [myConnCtx] {INSERT INTO DEPARTMENT VALUES
  ("X00","Operations 2","000030","E01",NULL)};
#sql [myConnCtx] {UPDATE DEPARTMENT
  SET MGRNO="000090" WHERE MGRNO="000030"};
#sql [myConnCtx] {DELETE FROM DEPARTMENT
  WHERE DEPTNO="X00"};
```

Ejecución de operaciones UPDATE y DELETE de posición en una aplicación SQLJ

Al igual que ocurre en las aplicaciones DB2 escritas en otros lenguajes, la ejecución de sentencias UPDATE y DELETE de posición en SQLJ es una extensión de la recuperación de filas a partir de la tabla de resultados.

Acerca de esta tarea

Los pasos básicos son:

Procedimiento

1. Declare el iterador.

   El iterador puede ser de posición o de nombre. Para las operaciones UPDATE o DELETE posicionadas, debe declarar el iterador como actualizable mediante una de las cláusulas siguientes o ambas:

   implements sqlj.runtime.ForUpdate
   Esta cláusula hace que la clase de iterador creada incluya métodos para utilizar iteradores actualizables. Esta cláusula es necesaria para los programas con operaciones UPDATE o DELETE de posición.

   with (updateColumns="*lista-columnas***")**
   Esta cláusula especifica una lista de las columnas, separadas por comas, de la tabla de resultados que serán actualizadas por el iterador. Esta cláusula es opcional.

   Es necesario declarar el iterador como `public`, por lo que es preciso seguir las normas de declaración y utilización de iteradores `public` en el mismo archivo o en archivos diferentes.

   Si declara el iterador en un archivo separado, cualquier archivo fuente SQLJ que pueda acceder al iterador e importe la clase generada puede recuperar datos y ejecutar sentencias UPDATE o DELETE de posición utilizando el iterador.

   El ID de autorización utilizado para ejecutar la sentencia UPDATE o DELETE de posición depende de si la sentencia se ejecuta de forma estática o dinámica. Si la sentencia se ejecuta estáticamente, el ID de autorización es el propietario del plan o paquete donde reside la sentencia. Si la sentencia se ejecuta de forma dinámica, el ID de autorización está determinado por la acción de DYNAMICRULES que esté en vigor. Para el IBM Data Server Driver para JDBC y SQLJ, el comportamiento es siempre DYNAMICRULES BIND.

2. Inhabilite la modalidad de confirmación automática (autocommit) para la conexión.

   Cuando la modalidad de confirmación automática está habilitada, se ejecuta una operación COMMIT cada vez que se ejecuta una sentencia UPDATE de posición, lo que provoca la destrucción del iterador a menos que el iterador tenga el atributo `with (holdability=true)`. Por tanto, es necesario desactivar la confirmación automática para evitar que se ejecuten operaciones COMMIT hasta que haya terminado de utilizar el iterador. Si desea que se ejecute una operación COMMIT después de cada actualización, una forma alternativa de impedir la destrucción del iterador después de cada operación COMMIT es declarar el iterador con el atributo `with (holdability=true)`.

3. Cree una instancia de la clase de iterador.

   Este paso es el mismo que para un iterador no actualizable.

4. Asigne la tabla de resultados de una sentencia SELECT a una instancia del iterador.

   Este paso es el mismo que para un iterador no actualizable. La sentencia SELECT no debe incluir una cláusula FOR UPDATE.

5. Recupere y actualice filas.

   Para un iterador de posición, ejecute las acciones siguientes en bucle:

a. Ejecute una sentencia FETCH en una cláusula ejecutable para obtener la fila actual.
b. Invoque el método PositionedIterator.endFetch para determinar si el iterador está apuntando a una fila de la tabla de resultados.
c. Si el iterador está apuntando a una fila de la tabla de resultados, ejecute una sentencia UPDATE... WHERE CURRENT OF *:objeto-iterador* de SQL en una cláusula ejecutable para actualizar las columnas de la fila actual. Ejecute una sentencia DELETE... WHERE CURRENT OF *:objeto-iterador* de SQL en una cláusula ejecutable para suprimir la fila actual.

Para un iterador de nombre, ejecute las acciones siguientes en bucle:

a. Invoque el método next para avanzar el iterador.
b. Determine si el iterador está apuntando a una fila de la tabla de resultados; para ello compruebe si next devuelve el valor `true`.
c. Ejecute una sentencia UPDATE... WHERE CURRENT OF *:objeto-iterador* de SQL en una cláusula ejecutable para actualizar las columnas de la fila actual. Ejecute una sentencia DELETE... WHERE CURRENT OF *objeto-iterador* de SQL en una cláusula ejecutable para suprimir la fila actual.

6. Cierre el iterador.

   Para ello utilice el método close.

Ejemplo

El código siguiente muestra cómo declarar un iterador de posición y utilizarlo para operaciones UPDATE de posición. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

En primer lugar, en un archivo individual, declare un iterador de posición `UpdByPos`, y especifique que desea utilizar el iterador para actualizar la columna `SALARY`:

```
import java.math.*;    // Importar esta clase para tipo de datos BigDecimal
#sql public iterator UpdByPos implements sqlj.runtime.ForUpdate       1
  with(updateColumns="SALARY") (String, BigDecimal);
```

Figura 27. Ejemplo de declaración de un iterador de posición para una sentencia UPDATE de posición

A continuación, en otro archivo, utilice `UpdByPos` para un UPDATE de posición, tal como se muestra en el fragmento de código siguiente:

```
import sqlj.runtime.*;  // Importar archivos para las API de SQLJ y JDBC
import java.sql.*;
import java.math.*;          // Importar esta clase para tipo de datos BigDecimal
import UpdByPos;             // Importar clase de iterador generada que se ha
                             // creado mediante la cláusula de declaración de
                             // iterador para UpdByName en otro archivo
#sql context HSCtx;          // Crear la clase de contexto de conexión HSCtx
public static void main (String args[])
{
  try {
    Class.forName("com.ibm.db2.jcc.DB2Driver");
  }
  catch (ClassNotFoundException e) {
    e.printStackTrace();
          }
  Connection HSjdbccon=
  DriverManager.getConnection("jdbc:db2:SANJOSE");
                     // Crear un objeto de conexión de JDBC
  HSjdbccon.setAutoCommit(false);
                     // Establecer autocommit en off para que las        2
                         // confirmaciones automáticas no destruyan el cursor
                         // entre actualizaciones
  HSCtx myConnCtx=new HSCtx(HSjdbccon);
                     // Crear un objeto de contexto de conexión
  UpdByPos upditer;  // Declarar objeto de iterador de la clase UpdByPos   3
  String empnum;     // Declarar variable de lenguaje principal para
  BigDecimal sal;    // recibir los valores de las columnas EMPNO y SALARY
  #sql [myConnCtx]
    upditer = {SELECT EMPNO, SALARY FROM EMPLOYEE                          4
      WHERE WORKDEPT='D11'};
                     // Asignar tabla de resultados a objeto de iterador
  #sql {FETCH :upditer INTO :empnum,:sal};                                 5a
                          // Avanzar cursor hasta la fila siguiente
  while (!upditer.endFetch())                                              5b
                          // Determinar si iterador apunta a una fila
  {
    #sql [myConnCtx] {UPDATE EMPLOYEE SET SALARY=SALARY*1.05
      WHERE CURRENT OF :upditer};                                          5c
                          // Ejecutar actualización de posición
    System.out.println("Actualizando fila para " + empnum);
    #sql {FETCH :upditer INTO :empnum,:sal};
                          // Avanzar cursor hasta la fila siguiente
  }
  upditer.close();        // Cerrar el iterador                           6
  #sql [myConnCtx] {COMMIT};
                          // Confirmar los cambios
  myConnCtx.close();      // Cerrar el contexto de conexión
}
```

Figura 28. Ejemplo de ejecución de una sentencia UPDATE de posición con un iterador de posición

El código siguiente muestra cómo declarar un iterador de nombre y utilizarlo para operaciones UPDATE de posición. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

En primer lugar, en un archivo individual, declare el iterador de nombre UpdByName, y especifique que desea utilizar el iterador para actualizar la columna SALARY:

```
import java.math.*;          // Importar esta clase para tipo de datos BigDecimal
#sql public iterator UpdByName implements sqlj.runtime.ForUpdate     1
  with(updateColumns="SALARY") (String EmpNo, BigDecimal Salary);
```

Figura 29. Ejemplo de declaración de un iterador con nombre para una sentencia UPDATE de posición

A continuación, en otro archivo, utilice UpdByName para un UPDATE de posición, tal como se muestra en el fragmento de código siguiente:

```
import sqlj.runtime.*;  // Importar archivos para las API de SQLJ y JDBC
import java.sql.*;
import java.math.*;          // Importar esta clase para tipo de datos BigDecimal
import UpdByName;            // Importar la clase de iterador generada que se
                             // creado mediante la cláusula de declaración de
                             // iterador para UpdByName en otro archivo
#sql context HSCtx;          // Crear la clase de contexto de conexión HSCtx
public static void main (String args[])
{
  try {
    Class.forName("com.ibm.db2.jcc.DB2Driver");
  }
  catch (ClassNotFoundException e) {
    e.printStackTrace();
          }
  Connection HSjdbccon=
  DriverManager.getConnection("jdbc:db2:SANJOSE");
                         // Crear un objeto de conexión de JDBC
  HSjdbccon.setAutoCommit(false);
                         // Establecer autocommit en off para que las        2
                         // confirmaciones automáticas no destruyan el cursor
                         // entre actualizaciones
  HSCtx myConnCtx=new HSCtx(HSjdbccon);
                         // Crear un objeto de contexto de conexión
  UpdByName upditer;                                                        3
                         // Declarar objeto de iterador de clase UpdByName
  String empnum;         // Declarar variable de leng. principal para recibir EmpNo
                             // recibir valores de la columna EmpNo
  #sql [myConnCtx]
    upditer = {SELECT EMPNO, SALARY FROM EMPLOYEE                           4
      WHERE WORKDEPT='D11'};
                             // Asignar tabla de resultados a objeto de iterador
  while (upditer.next())                                                    5a,5b
                             // Avanzar cursor hasta la fila siguiente
                             // Determinar si iterador apunta a una fila
  {
    empnum = upditer.EmpNo();  // Obtener n.º empleado desde fila actual
    #sql [myConnCtx]
      {UPDATE EMPLOYEE SET SALARY=SALARY*1.05
        WHERE CURRENT OF :upditer};                                         5c
                             // Ejecutar actualización de posición
    System.out.println("Actualizando fila para " + empnum);
  }
  upditer.close();           // Cerrar el iterador                         6
  #sql [myConnCtx] {COMMIT};
                             // Confirmar los cambios
  myConnCtx.close();         // Cerrar el contexto de conexión
}
```

Figura 30. Ejemplo de ejecución de una sentencia UPDATE de posición con un iterador con nombre

Uso de iteradores pasados como variables en operaciones UPDATE o DELETE de posición de una aplicación SQLJ

SQLJ permite pasar iteradores entre métodos en calidad de variables.

Un iterador que se utilice para una sentencia UPDATE o DELETE de posición sólo se puede identificar durante el tiempo de ejecución. La misma sentencia UPDATE o DELETE de posición de SQLJ se puede utilizar con iteradores diferentes durante la ejecución. Si especifica el valor YES para -staticpositioned cuando personaliza su aplicación SQLJ como parte del proceso de preparación del programa, el personalizador de SQLJ prepara sentencias UPDATE o DELETE de posición para su ejecución estática. En este caso, el personalizador debe determinar qué iteradores pertenecen a cada sentencia UPDATE o DELETE de posición. Para ello, el personalizador de SQLJ asocia tipos de datos de iterador con tipos de datos de las sentencias UPDATE o DELETE. Sin embargo, no existe una correspondencia unívoca entre las tablas de las sentencias UPDATE o DELETE y las clases de iterador, el personalizador de SQLJ no puede determinar exactamente qué iterador pertenece a cada sentencia UPDATE o DELETE. En este caso, el personalizador de SQLJ debe asociar arbitrariamente iteradores con sentencias UPDATE o DELETE, lo cual puede a veces producir errores de SQL. Esto se muestra en los fragmentos de código siguientes.

```
#sql iterator GeneralIter implements sqlj.runtime.ForUpdate
  ( String );

  public static void main ( String args[] )
  {
...
    GeneralIter iter1 = null;
    #sql [ctxt] iter1 = { SELECT CHAR_COL1 FROM TABLE1 };

    GeneralIter iter2 = null;
    #sql [ctxt] iter2 = { SELECT CHAR_COL2 FROM TABLE2 };
...

    doUpdate ( iter1 );
  }

  public static void doUpdate ( GeneralIter iter )
  {
    #sql [ctxt] { UPDATE TABLE1 ... WHERE CURRENT OF :iter };
  }
```

Figura 31. UPDATE de posición estática que ejecuta incorrectamente

En este ejemplo sólo se declara un iterador. Se declaran dos instancias de ese iterador, y cada una de ellas se asocia a una sentencia SELECT diferente que recupera datos de una tabla diferente. Durante la personalización y vinculación con -staticpositioned YES, SQLJ crea dos sentencias DECLARE CURSOR, una para cada sentencia SELECT e intenta vincular una sentencia UPDATE para cada cursor. Sin embargo, el proceso de vinculación no se consigue con SQLCODE -509 cuando `UPDATE TABLE1 ... WHERE CURRENT OF :iter` está vinculado para el cursor para `SELECT CHAR_COL2 FROM TABLE2`, ya que la tabla de UPDATE no coincide con la tabla del cursor.

Puede evitar un error de tiempo de vinculación para un programa como el de la Figura 31 especificando la opción de vinculación SQLERROR(CONTINUE). Pero esta técnica tiene el inconveniente de que hace que el gestor de bases de datos DB2 cree un paquete, sin importar los errores de SQL existentes en el programa. Una técnica mejor consiste en grabar el programa de forma que exista una correspondencia unívoca entre las tablas de las sentencias UPDATE o DELETE de posición y las clases de iterador. La Figura 32 en la página 165 muestra un ejemplo de cómo hacerlo.

```
#sql iterator Table2Iter(String);
#sql iterator Table1Iter(String);
  public static void main ( String args[] )
  {
...
    Table2Iter iter2 = null;
    #sql [ctxt] iter2 = { SELECT CHAR_COL2 FROM TABLE2 };

    Table1Iter iter1 = null;
    #sql [ctxt] iter1 = { SELECT CHAR_COL1 FROM TABLE1 };
...

    doUpdate(iter1);

  }

 public static void doUpdate ( Table1Iter iter )
  {
    ...
    #sql [ctxt] { UPDATE TABLE1 ... WHERE CURRENT OF :iter };
    ...
  }
 public static void doUpdate ( Table2Iter iter )
  {
    ...
    #sql [ctxt] { UPDATE TABLE2 ... WHERE CURRENT OF :iter };
    ...
  }
```

Figura 32. Sentencia UPDATE estática de posición que se ejecuta satisfactoriamente

Con esta forma de codificación, cada clase de iterador se asocia a una sola tabla. Por tanto, el proceso de vinculación de DB2 puede siempre asociar la sentencia UPDATE de posición con un iterador válido.

Realización de actualizaciones por lotes en aplicaciones SQLJ

IBM Data Server Driver para JDBC y SQLJ soporta la realización de actualizaciones por lotes en SQLJ. En las actualizaciones de proceso por lotes, en lugar de actualizar filas de una tabla de forma individual, puede hacer que SQLJ ejecute un grupo de actualizaciones al mismo tiempo.

Acerca de esta tarea

Puede incluir los tipos siguientes de sentencias en una actualización por lotes:
- Sentencias INSERT, UPDATE o DELETE o MERGE buscadas
- Sentencias CREATE, ALTER, DROP, GRANT y REVOKE
- Sentencias CALL con parámetros de entrada solamente

A diferencia de JDBC, SQLJ permite utilizar lotes heterogéneos que contienen sentencias con parámetros de entrada o expresiones de lenguaje principal. Por consiguiente, puede combinar cualquiera de los elementos siguientes en un proceso por lotes SQLJ:

- Instancias de la misma sentencia
- Sentencias diferentes
- Sentencias con números de parámetros de entrada o expresiones de lenguaje principal diferentes
- Sentencias con tipos de datos diferentes para parámetros de entrada o expresiones de lenguaje principal
- Sentencias sin parámetros de entrada ni expresiones de lenguaje principal

Para todos los casos excepto los lotes homogéneos de sentencias INSERT, cuando se produce un error durante la ejecución de una sentencia de un proceso por lotes, se ejecutan las sentencias y se emite una excepción BatchUpdateException una vez ejecutadas todas las sentencias del proceso por lotes.

Para los lotes homogéneos de sentencias INSERT, el comportamiento es el siguiente:

- Si establece atomicMultiRowInsert en DB2BaseDataSource.YES (1) al ejecutar db2sqljcustomize y el servidor de datos de destino es DB2 para z/OS, cuando se produce un error durante la ejecución de una sentencia INSERT en un lote, no se ejecutan las sentencias restantes y se genera una excepción BatchUpdateException.
- Si no establece atomicMultiRowInsert en DB2BaseDataSource.YES (1) al ejecutar db2sqljcustomize o el servidor de datos de destino no es DB2 para z/OS, cuando se produce un error durante la ejecución de una sentencia INSERT en un lote, se ejecutan las sentencias restantes y se genera una excepción BatchUpdateException después de que se hayan ejecutado todas las sentencias del lote.

Para obtener información sobre avisos, utilice el método ExecutionContext.getWarnings en el ExecutionContext utilizado para enviar las sentencias que se van a procesar por lotes. Luego puede obtener una descripción de error, el estado de SQL y el código de error correspondientes a cada objeto SQLWarning.

Cuando un proceso por lotes se ejecuta implícitamente debido a que el programa contiene una sentencia que no se puede añadir a dicho proceso por lotes, el proceso por lotes se ejecutará antes de procesar la nueva sentencia. Si se produce un error al ejecutar el proceso por lotes, la sentencia que provocó la ejecución del proceso por lotes no se ejecutará.

Estos son los pasos básicos para crear, ejecutar y suprimir un lote de sentencias:

Procedimiento

1. Inhabilite `AutoCommit` (confirmación automática) para la conexión.

   Realice esto para poder controlar si se deben confirmar los cambios realizados en sentencias ya ejecutadas cuando se produce un error durante la ejecución de proceso por lotes.
2. Obtenga un contexto de ejecución.

   Todas las sentencias que se ejecutan en un lote deben utilizar este contexto de ejecución.
3. Invoque el método ExecutionContext.setBatching(true) para crear un lote.

   Las subsiguientes sentencias procesables por lotes que están asociadas al contexto de ejecución creado en el paso 2 se añaden al lote para su ejecución posterior.

   Si desea procesar por lotes conjuntos de sentencias que no son compatibles respecto al proceso por lotes en paralelo, debe crear un contexto de ejecución para cada conjunto de sentencias compatibles respecto al proceso por lotes.
4. Incluya cláusulas ejecutables de SQLJ para las sentencias de SQL que desee procesar por lotes.

   Estas cláusulas deben incluir el contexto de ejecución que creó en el paso 2.

Si una cláusula ejecutable de SQLJ tiene parámetros de entrada o expresiones de lenguaje principal, puede incluir la sentencia en el lote varias veces con valores diferentes para los parámetros de entrada o expresiones de lenguaje principal.

Para determinar si una sentencia se añadió a un lote existente, si era la primera sentencia de un nuevo lote, o si se ejecutó dentro o fuera de un lote, invoque el método ExecutionContext.getUpdateCount. Este método devuelve uno de los valores siguientes:

ExecutionContext.ADD_BATCH_COUNT
Se devuelve esta constante si la sentencia se añadió a un lote existente.

ExecutionContext.NEW_BATCH_COUNT
Se devuelve esta constante si la sentencia era la primera sentencia de un nuevo lote.

ExecutionContext.EXEC_BATCH_COUNT
Se devuelve esta constante si la sentencia formaba parte de un lote que se ejecutó.

Otro valor entero
Este valor es el número de filas que fueron actualizadas por la sentencia. Se devuelve este valor si se ejecutó la sentencia en lugar de añadirla a un lote.

5. Ejecute el lote explícita o implícitamente.
   - Invoque el método ExecutionContext.executeBatch para ejecutar el lote explícitamente.

     executeBatch devuelve una matriz entera que contiene el número de filas que fueron actualizadas por cada sentencia del lote. El orden de los elementos de la matriz corresponde al orden en el que se añadieron las sentencias al lote.
   - Como alternativa, un lote se ejecuta implícitamente en las condiciones siguientes:
     – Cuando incluye en su programa una sentencia procesable por lotes que no es compatible con sentencias ya existentes en el lote. En este caso, SQLJ ejecuta las sentencias que ya existen en el lote y crea un nuevo lote donde se incluye la sentencia incompatible.
     – Cuando incluye en su programa una sentencia que no es ejecutable por lotes. En este caso, SQLJ ejecuta las sentencias que ya existen en el lote. SQLJ también ejecuta la sentencia que no es procesable por lotes.
     – Cuando después de invocar el método `ExecutionContext.setBatchLimit(n)`, añade una sentencia al lote que hace que el número de sentencias del lote sea igual o mayor que *n*. *n* puede tener uno de los valores siguientes:

       ExecutionContext.UNLIMITED_BATCH
       Esta constante indica que la ejecución implícita solo se produce cuando SQLJ encuentra una sentencia que es procesable por lotes pero incompatible, o que no es procesable por lotes. Establecer este valor es lo mismo que no invocar setBatchLimit.

       ExecutionContext.AUTO_BATCH
       Esta constante indica que la ejecución implícita se produce cuando el número de sentencias del lote alcanza un valor definido por SQLJ.

       Entero positivo
       Cuando el número de sentencias añadidas al lote alcanza este valor, SQLJ ejecuta el lote implícitamente. Sin embargo, el proceso por lotes debería ejecutarse antes de que estas muchas sentencias se hayan

añadido en el caso de que SQLJ encuentre una sentencia que se pueda procesar por lotes pero que sea incompatible o bien que la sentencia no se pueda procesar por lotes.

Para determinar el número de filas que fueron actualizadas por las sentencias de un lote que se ejecutó implícitamente, invoque el método ExecutionContext.getBatchUpdateCounts. getBatchUpdateCounts devuelve una matriz entera que contiene el número de filas que fueron actualizadas por cada sentencia del lote. El orden de los elementos de la matriz corresponde al orden en el que se añadieron las sentencias al lote. Cada elemento de la matriz puede ser uno de los valores siguientes:

-2 Este valor indica que la sentencia de SQL se ejecutó satisfactoriamente, pero no se pudo determinar el número de filas que fueron actualizadas.

-3 Este valor indica que la sentencia de SQL falló.

Otro valor entero
Este valor es el número de filas que fueron actualizadas por la sentencia.

6. Opcionalmente, una vez añadidas todas las sentencias al lote, puede inhabilitar el proceso por lotes.

   Para ello invoque el método ExecutionContext.setBatching(false). Cuando inhabilita el proceso por lotes, puede todavía ejecutar el lote implícita o explícitamente, pero no se añaden más sentencias al lote. Inhabilitar el proceso por lotes es útil cuando ya existe un lote y desea ejecutar una sentencia compatible de proceso por lotes, en lugar de añadirla al lote.

   Si desea eliminar un lote sin ejecutarlo, invoque el método ExecutionContext.cancel.

7. Si la ejecución por lotes era implícita, realice una ejecución final, explícita de executeBatch para asegurarse de que se hayan ejecutado todas las sentencias.

Ejemplo

El ejemplo siguiente muestra el proceso por lotes de las sentencias UPDATE. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
#sql iterator GetMgr(String);  // Declarar iterador de posición
...
{
  GetMgr deptiter;            // Declarar objeto de la clase GetMgr
  String mgrnum = null;       // Declarar variable de lenguaje principal
                              // para número de director de departamento
  int raise = 400;                  // Declarar el importe del aumento
  int currentSalary;          // Declarar salario actual
  String url, username, password;   // Declarar url, ID de usuario, contraseña
  ...
  TestContext c1 = new TestContext (url, username, password, false);  1
  ExecutionContext ec = new ExecutionContext();                       2
  ec.setBatching(true);                                               3

  #sql [c1] deptiter =
    {SELECT MGRNO FROM DEPARTMENT};
                                    // Asignar tabla de resultados de SELECT
                                    // al objeto iterador deptiter
  #sql {FETCH :deptiter INTO :mgrnum};
                                    // Recuperar el primer número del gestor
  while (!deptiter.endFetch()) {    // Comprobar si FETCH ha devuelto una fila
    #sql [c1]
      {SELECT SALARY INTO :currentSalary FROM EMPLOYEE
        WHERE EMPNO=:mgrnum};
    #sql [c1, ec]                                                     4
```

```
      {UPDATE EMPLOYEE SET SALARY=:(currentSalary+raise)
        WHERE EMPNO=:mgrnum};
    #sql {FETCH :deptiter INTO :mgrnum };
                                    // Obtener la fila siguiente
  }
  ec.executeBatch();                                                    5
  ec.setBatching(false);                                                6
  #sql [c1] {COMMIT};
  deptiter.close();               // Cerrar el iterador
  c1.close();                     // Cerrar la conexión
}
```

El ejemplo siguiente muestra el proceso por lotes de las sentencias INSERT. Supongamos que ATOMICTBL se ha definido de la siguiente manera:

```
CREATE TABLE ATOMICTBL(
 INTCOL INTEGER NOT NULL UNIQUE,
 CHARCOL VARCHAR(10))
```

Supongamos también que la tabla ya tiene una fila con los valores 2 y "val2". A causa de la restricción de exclusividad en INTCOL, cuando se ejecuta el código siguiente, la segunda sentencia INSERT del lote genera un error.

Si el servidor de datos de destino es DB2 para z/OS y esta aplicación se ha personalizado sin que atomicMultiRowInsert esté establecido en DB2BaseDataSource.YES, las inserciones de proceso por lotes son no atómicas, de modo que el primer conjunto de valores se inserta en la tabla. Sin embargo, si la aplicación se ha personalizado con atomicMultiRowInsert establecido en DB2BaseDataSource.YES, las inserciones de proceso por lotes son atómicas, de modo que no se inserta el primer conjunto de valores.

Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
 ...
TestContext ctx = new TestContext (url, username, password, false);  1
ctx.getExecutionContext().setBatching(true);                         2,3
try {
  for (int i = 1; i<= 2; ++i) {
   if (i == 1) {
    intVar = 3;
    strVar = "val1";
   {
   if (i == 2) {
    intVar = 1;
    strVar = "val2";
   }
   #sql [ctx] {INSERT INTO ATOMICTBL values(:intVar, :strVar)};       4
  }
  int[] counts = ctx.getExecutionContext().executeBatch();           5
  for (int i = 0; i<counts.length; ++i) {
    System.out.println(" count[" + i + "]:" + counts[i]);
  }
}
catch (SQLException e) {
  System.out.println(" Exception Caught: " + e.getMessage());
   SQLException excp = null;
  if (e instanceof SQLException)
  {
    System.out.println(" SQLCode: " + ((SQLException)e).getErrorCode() + "
      Message: " + e.getMessage() );
    excp = ((SQLException)e).getNextException();
    while (  excp != null ) {
      System.out.println(" SQLCode: " + ((SQLException)excp).getErrorCode() +
        " Message: " + excp.getMessage() );
```

```
      excp = excp.getNextException();
    }
  }
}
```

Recuperación de datos en aplicaciones SQLJ

Las aplicaciones SQLJ utilizan un *iterador de conjunto de resultados* para recuperar conjuntos de resultados. Al igual que un cursor, un iterador de conjunto de resultados puede ser desplazable o no desplazable.

Al igual que ocurre en aplicaciones DB2 escritas en otros lenguajes, si desea recuperar una fila individual de una tabla en una aplicación SQLJ, puede escribir una sentencia SELECT INTO con una cláusula WHERE que define una tabla de resultados que contiene solamente esa fila:

```
#sql [myConnCtx] {SELECT DEPTNO INTO :hvdeptno
  FROM DEPARTMENT WHERE DEPTNAME="OPERATIONS"};
```

Sin embargo, la mayoría de las sentencias SELECT que utiliza producen tablas de resultados que contienen muchas filas. En las aplicaciones DB2 escritas en otros lenguajes, utiliza un cursor para seleccionar filas individuales de la tabla de resultados. Ese cursor puede ser no desplazable, lo que significa que cuando lo utiliza para recuperar filas, desplaza el cursor secuencialmente desde el principio de la tabla de resultados hasta el final. Como alternativa, el cursor puede ser desplazable, lo que significa que cuando lo utiliza para recuperar filas, puede desplazar el cursor hacia delante y atrás o situarlo en una fila cualquiera de la tabla de resultados.

El presente tema describe cómo utilizar iteradores no desplazables. Para obtener información sobre la utilización de iteradores desplazables, consulte "Utilización de iteradores desplazables en una aplicación SQLJ".

Un iterador de conjunto de resultados es un objeto Java que se utiliza para recuperar filas de una tabla de resultados. A diferencia de un cursor, un iterador de conjunto de resultados se puede pasar como parámetro a un método.

Estos son los pasos básicos para utilizar un iterador de conjunto de resultados:
1. Declare el iterador, con lo que se creará una clase de iterador
2. Defina una instancia de la clase de iterador.
3. Asigne la tabla de resultados de una sentencia SELECT a una instancia del iterador.
4. Recupere filas.
5. Cierre el iterador.

Existen dos tipos de iteradores: *iteradores de posición* e *iteradores de nombre*. Los iteradores de posición amplían la interfaz sqlj.runtime.PositionedIterator. Los iteradores de posición identifican las columnas de una tabla de resultados por la posición que ocupan en la tabla de resultados. Los iteradores de nombre amplían la interfaz sqlj.runtime.NamedIterator. Los iteradores de nombre identifican las columnas de la tabla de resultados por su nombre.

Utilización de un iterador de nombre en una aplicación SQLJ

Utilice un iterador con nombre para especificar por su nombre cada columna de una tabla de resultados.

Acerca de esta tarea

Estos son los pasos para utilizar un iterador de nombre:

Procedimiento

1. Declare el iterador.

   Utilice una *cláusula de declaración de iterador* para declarar un iterador de conjunto de resultados. Esto hace que se cree una clase de iterador que tiene el mismo nombre que el iterador. Para un iterador de nombre, la cláusula de declaración de iterador especifica la información siguiente:

   - El nombre del iterador
   - Una lista de nombres de columnas y tipos de datos Java
   - Información para una declaración de clase Java, tal como la indicación de si el iterador es `public` o `static`
   - Un conjunto de atributos, tales como si el iterador se puede retener o si sus columnas se pueden actualizar.

   Cuando declara un iterador de nombre para una consulta, especifica nombres para cada columna del iterador. Estos nombres deben coincidir con los nombres de las columnas de la tabla de resultados para la consulta. Un nombre de columna de iterador y un nombre de columna de tabla de resultados que sólo difieran en el aspecto de que contengan mayúsculas o minúsculas se consideran nombres coincidentes. La clase de iterador de nombre generada por la cláusula de declaración de iterador contiene *métodos accesores*. Existe un método accesor para cada columna del iterador. Cada método accesor tiene el mismo nombre que la columna correspondiente del iterador. Los métodos accesores se utilizan para recuperar datos contenidos en columnas de la tabla de resultados.

   Es necesario que en los iteradores especifique tipos de datos Java que coincidan estrechamente con los correspondientes tipos de datos de columna de DB2. Consulte "Tipos de datos Java, JDBC y SQL" para obtener una lista de las mejores correspondencias entre los tipos de datos Java y los tipos de datos DB2.

   Puede declarar un iterador de varias maneras. Sin embargo, debido a que cada iterador está asociado a una clase Java, cuando declare un iterador, la clase subyacente debe cumplir las normas de Java. Por ejemplo, los iteradores que contienen una *cláusula-with* se deben declarar como `public`. Por lo tanto, si es necesario que un iterador sea `public`, sólo se puede declarar donde se admita una clase `public`. La lista siguiente describe algunos métodos alternativos para declarar un iterador:

   - Como `public`, en un archivo fuente separado

     Este método le permite utilizar la declaración de iterador en otros módulos de código, y proporciona un iterador que es efectivo para todas las aplicaciones SQLJ. Además, no existen las cuestiones derivadas de tener otras clases de nivel superior o clases `public` en el mismo archivo fuente.
   - Como clase de nivel superior en un archivo fuente que contiene otras definiciones de clases de nivel superior

     Java permite una sola clase pública, de nivel superior, en un módulo de código. Por tanto, si necesita declarar el iterador como público, tal como cuando el iterador incluye una cláusula with, ninguna otra clase contenida en el módulo de código puede estar declarada como pública.
   - Como clase estática anidada dentro de otra clase

     Esta alternativa le permite combinar la declaración de iterador con otras declaraciones de clases en el mismo archivo fuente, declarar el iterador y otras clases como públicos y hacer que la clase de iterador sea visible para

otros módulos de código o paquetes. Si embargo, si especifica el iterador desde fuera de la clase anidadora debe calificar por completo el nombre de iterador con el nombre de la clase anidadora.

- Como clase interna dentro de otra clase

  Cuando declara un iterador de esta manera, puede crear una instancia del iterador solo dentro de una instancia de la clase anidadora. Sin embargo, puede declarar como públicos el iterador y otras clases contenidas en el archivo.

  No puede convertir un ResultSet de JDBC en un iterador si el iterador está declarado como clase interna. Esta restricción no es aplicable si el iterador está declarado como clase anidada estática. Consulte "Utilizar SQLJ y JDBC en la misma aplicación" para obtener más información sobre la conversión de un ResultSet en un iterador.

2. Cree una instancia de la clase de iterador.

   Debe declarar un objeto de la clase de iterador de nombre para recuperar filas de una tabla de resultados.

3. Asigne la tabla de resultados de una sentencia SELECT a una instancia del iterador.

   Para asignar la tabla de resultados de una sentencia SELECT a un iterador, utilice una *cláusula de asignación* de SQLJ. Este es el formato de la cláusula de asignación para un iterador de nombre:

   ```
   #sql cláusula-contexto objeto-iterador={sentencia-select};
   ```

   Consulte "Cláusula de asignación de SQLJ" y "Cláusula de contexto de SQLJ" para obtener más información.

4. Recupere filas.

   Para ello invoque métodos accesores en un bucle. Los métodos accesores tienen los mismos nombres que las columnas correspondientes del iterador, y carecen de parámetros. Un método accesor devuelve el valor de la columna correspondiente de la fila actual de la tabla de resultados. Utilice el método NamedIterator.next() para avanzar el cursor por la tabla de resultados.

   Para determinar si ha recuperado todas las filas, examine el valor devuelto al invocar el método next. next devuelve un valor de tipo `boolean` igual a `false` si no existe ninguna fila siguiente.

5. Cierre el iterador.

   Para ello utilice el método NamedIterator.close.

Ejemplo

El código siguiente muestra cómo declarar y utilizar un iterador de nombre. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
#sql  iterator ByName(String LastName, Date HireDate);                  1
                                  // Declarar iterador de nombre ByName
{
  ...
  ByName nameiter;                // Declarar objeto de clase ByName       2
  #sql [ctxt]
 nameiter={SELECT LASTNAME, HIREDATE FROM EMPLOYEE};                      3
                                  // Asignar tabla de resultados de SELECT
                                  // al objeto iterador nameiter
  while (nameiter.next())         // Mover el iterador por la tabla de     4
                                  // resultados y evaluar si se han
                                  // recuperado todas las filas
  {
    System.out.println( nameiter.LastName() + " se contrató el "
      + nameiter.HireDate());     // Usar métodos de acceso LastName y
                                  // HireDate para recuperar valores de columnas
  }
  nameiter.close();               // Cerrar el iterador                    5
}
```

Figura 33. Ejemplo de utilización de un iterador con nombre

Utilización de un iterador de posición en una aplicación SQLJ

Utilice un iterador de posición para especificar columnas de una tabla de resultados indicando la posición de la columna en el conjunto de resultados.

Acerca de esta tarea

Estos son los pasos para utilizar un iterador de posición:

Procedimiento

1. Declare el iterador.

   Utilice una *cláusula de declaración de iterador* para declarar un iterador de conjunto de resultados. Esto hace que se cree una clase de iterador que tiene el mismo nombre y atributos que el iterador. Para un iterador de posición, la cláusula de declaración de iterador especifica la información siguiente:

   - El nombre del iterador
   - Una lista de tipos de datos Java
   - Información para una declaración de clase Java, tal como la indicación de si el iterador es `public` o `static`
   - Un conjunto de atributos, tales como si el iterador se puede retener o si sus columnas se pueden actualizar.

   Las declaraciones de tipos de datos representan columnas de la tabla de resultados y se especifican como columnas del iterador del conjunto de resultados. Las columnas del iterador del conjunto de resultados se corresponden con las columnas de la tabla de resultados, en el orden de izquierda a derecha. Por ejemplo, si una cláusula de declaración de iterador tiene dos declaraciones de tipos de datos, la primera declaración corresponde a la primera columna de la tabla de resultados, y la segunda declaración corresponde a la segunda columna de la tabla de resultados.

   Es necesario que en los iteradores especifique tipos de datos Java que coincidan estrechamente con los correspondientes tipos de datos de columna de DB2. Consulte "Tipos de datos Java, JDBC y SQL" para obtener una lista de las mejores correspondencias entre los tipos de datos Java y los tipos de datos DB2.

Puede declarar un iterador de varias maneras. Sin embargo, debido a que cada iterador está asociado a una clase Java, cuando declare un iterador, la clase subyacente debe cumplir las normas de Java. Por ejemplo, los iteradores que contienen una *cláusula-with* se deben declarar como `public`. Por lo tanto, si es necesario que un iterador sea `public`, sólo se puede declarar donde se admita una clase `public`. La lista siguiente describe algunos métodos alternativos para declarar un iterador:

- Como `public`, en un archivo fuente separado

  Este es el método más versátil de declarar un iterador. Este método le permite utilizar la declaración de iterador en otros módulos de código, y proporciona un iterador que es efectivo para todas las aplicaciones SQLJ. Además, no existen las cuestiones derivadas de tener otras clases de nivel superior o clases `public` en el mismo archivo fuente.
- Como clase de nivel superior en un archivo fuente que contiene otras definiciones de clases de nivel superior

  Java permite una sola clase pública, de nivel superior, en un módulo de código. Por tanto, si necesita declarar el iterador como público, tal como cuando el iterador incluye una cláusula with, ninguna otra clase contenida en el módulo de código puede estar declarada como pública.
- Como clase estática anidada dentro de otra clase

  Esta alternativa le permite combinar la declaración de iterador con otras declaraciones de clases en el mismo archivo fuente, declarar el iterador y otras clases como públicos y hacer que la clase de iterador sea visible desde otros módulos de código o paquetes. Si embargo, si especifica el iterador desde fuera de la clase anidadora debe calificar por completo el nombre de iterador con el nombre de la clase anidadora.
- Como clase interna dentro de otra clase

  Cuando declara un iterador de esta manera, puede crear una instancia del iterador solo dentro de una instancia de la clase anidadora. Sin embargo, puede declarar como públicos el iterador y otras clases contenidas en el archivo.

  No puede convertir un ResultSet de JDBC en un iterador si el iterador está declarado como clase interna. Esta restricción no es aplicable si el iterador está declarado como clase anidada estática. Consulte "Utilizar SQLJ y JDBC en la misma aplicación" para obtener más información sobre la conversión de un ResultSet en un iterador.

2. Cree una instancia de la clase de iterador.

   Debe declarar un objeto de la clase de iterador de posición para recuperar filas de una tabla de resultados.

3. Asigne la tabla de resultados de una sentencia SELECT a una instancia del iterador.

   Para asignar la tabla de resultados de una sentencia SELECT a un iterador, utilice una *cláusula de asignación* de SQLJ. Este es el formato de la cláusula de asignación para un iterador de posición:

   ```
   #sql cláusula-contexto objeto-iterador={sentencia-select};
   ```

4. Recupere filas.

   Para ello ejecute en bucle sentencias FETCH en cláusulas ejecutables. Las sentencias FETCH son iguales a las utilizadas en otros lenguajes de programación.

Para determinar si ha recuperado todas las filas, invoque el método PositionedIterator.endFetch después de cada FETCH. endFetch devuelve un valor de tipo `boolean` igual a `true` si FETCH falló porque no había filas para recuperar.

5. Cierre el iterador.

   Para ello utilice el método PositionedIterator.close.

Ejemplo

El código siguiente muestra cómo declarar y utilizar un iterador de posición. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
#sql iterator ByPos(String,Date); // Declarar iterador de posición ByPos  1
{
  ...
  ByPos positer;              // Declarar objeto de clase ByPos           2
  String name = null;        // Declarar variables de lenguaje principal
  Date hrdate;
  #sql [ctxt] positer =
    {SELECT LASTNAME, HIREDATE FROM EMPLOYEE};                            3
                                    // Asignar tabla de resultados de SELECT
                                    // al objeto iterador positer
  #sql {FETCH :positer INTO :name, :hrdate };                             4
                                    // Recuperar primera fila
  while (!positer.endFetch())   // Determinar si FETCH ha devuelto una fila
  { System.out.println(name + " fue contratado en " +
      hrdate);
    #sql {FETCH :positer INTO :name, :hrdate };
                                    // Obtener la fila siguiente
  }
  positer.close();                  // Cerrar el iterador                  5
}
```

Figura 34. Ejemplo de utilización de un iterador de posición

Varios iteradores abiertos para una misma sentencia de SQL en una aplicación SQLJ

Con IBM Data Server Driver para JDBC y SQLJ, la aplicación puede tener varios iteradores abiertos a la vez para una única sentencia de SQL de una aplicación SQLJ. Esta característica le permite efectuar una operación en una tabla utilizando un iterador mientras ejecuta una operación diferente en la misma tabla utilizando otro iterador.

Cuanto utilice simultáneamente varios iteradores abiertos en una aplicación, es conveniente que cierre los iteradores que ya no necesite, a fin de evitar un consumo excesivo de espacio de almacenamiento en el almacenamiento dinámico de Java.

Los ejemplos siguientes muestran la ejecución de unas mismas operaciones en una tabla con y sin operadores abiertos simultáneamente para una misma sentencia de SQL. Estos ejemplos utilizan la siguiente declaración de iterador:

```
import java.math.*;
#sql public iterator  MultiIter(String EmpNo, BigDecimal Salary);
```

Si no se utiliza la capacidad de tener abiertos a la vez varios iteradores para una misma sentencia de SQL, si desea seleccionar valores de empleado y salario para un número de empleado determinado, debe definir una sentencia de SQL diferente

para cada número de empleado, tal como muestra la Figura 35.

```
MultiIter iter1 = null;       // Instancia de iterador para
                              // recuperar datos para el primer empleado
String EmpNo1 = "000100";     // Número de empleado del primer empleado
#sql [ctx] iter1 =
  {SELECT EMPNO, SALARY FROM EMPLOYEE WHERE EMPNO = :EmpNo1};
                                   // Asignar tabla de resultados a primer iterador
MultiIter iter2 = null;       // Instancia de iterador para recuperar
                              // datos para el segundo empleado
String EmpNo2 = "000200";     // Número de empleado del segundo empleado
#sql [ctx] iter2 =
  {SELECT EMPNO, SALARY FROM EMPLOYEE WHERE EMPNO = :EmpNo2};
                                   // Asignar tabla de resultados a segundo iterador
// Procesar con iter1
// Procesar con iter2
iter1.close();                  // Cerrar los iteradores
iter2.close();
```

Figura 35. Ejemplo de operaciones de tabla simultáneas utilizando iteradores con sentencias de SQL diferentes

La Figura 36 muestra cómo ejecutar las mismas operaciones cuando existe la capacidad de tener abiertos a la vez varios iteradores para una misma sentencia de SQL.

```
...
MultiIter iter1 = openIter("000100"); // Invocar openIter para asignar
                                      // la tabla de resultados
                                      // (para el empleado 100) al primer
                                      // iterador
MultiIter iter2 = openIter("000200"); // Invocar openIter para asignar la tabla
                                      // de resultados al segundo iterador
                                      // iter1 permanece abierto cuando
                                      // se abre iter2
// Procesar con iter1
// Procesar con iter2
...
iter1.close();                 // Cerrar los iteradores
iter2.close();
...
public MultiIter openIter(String EmpNo)
                                      // Método para asignar una tabla de
                               // resultados a una instancia de iterador
{
  MultiIter iter;
  #sql [ctxt] iter =
    {SELECT EMPNO, SALARY FROM EMPLOYEE WHERE EMPNO = :EmpNo};
  return iter;                 // El método devuelve una instancia de iterador
}
```

Figura 36. Ejemplo de operaciones de tabla simultáneas utilizando iteradores con la misma sentencia de SQL

Uso de varias instancias abiertas de un iterador en una aplicación SQLJ

Pueden estar abiertas simultáneamente varias instancias de un iterador en una misma aplicación SQLJ. Una utilización de esta capacidad es abrir varias instancias de un iterador que hace uso de expresiones de lenguaje principal. Cada instancia puede utilizar un conjunto diferente de valores para una expresión de lenguaje principal.

El ejemplo siguiente muestra una aplicación con dos instancias abiertas simultáneamente de un iterador.

```
...
ResultSet myFunc(String empid)  // Método para abrir un iterador y
                                // obtener un conjunto de resultados
{
  MyIter iter;
  #sql iter = {SELECT * FROM EMPLOYEE WHERE EMPNO = :empid};
  return iter.getResultSet();
}

// Una aplicación puede invocar este método para obtener un conjunto
// de resultados para cada ID de empleado. La aplicación puede
// procesar cada conjunto de resultados por separado.
...
ResultSet rs1 = myFunc("000100");  // Obtener registro del ID de empleado 000100
...
ResultSet rs2 = myFunc("000200");  // Obtener registro del ID de empleado 000200
```

Figura 37. Ejemplo de apertura de más de una instancia de un iterador en una misma aplicación

Al igual que ocurre con cualquier otro iterador, es necesario cerrar este iterador cuando termine de utilizarlo para evitar un uso excesivo de espacio de almacenamiento.

Utilización de iteradores desplazables en una aplicación SQLJ

Además de avanzar fila a fila por una tabla de resultados, puede desear ir hacia atrás o directamente a una fila determinada. IBM Data Server Driver para JDBC y SQLJ proporciona esta capacidad.

Acerca de esta tarea

Un iterador en el que se puede desplazar hacia delante, hacia atrás o hasta una fila determinada se denomina *iterador desplazable*. Un iterador desplazable de SQLJ equivale a la tabla de resultados de un cursor de base de datos que está declarado como SCROLL.

Al igual que un cursor desplazable, un iterador desplazable puede ser *sensible* o *insensible*. Un iterador desplazable sensible puede ser *estático* o *dinámico*. Insensible significa que los cambios hechos en la tabla subyacente después de abrir el iterador no son visibles para el iterador. Los iteradores insensibles son de solo lectura. Sensible significa que los cambios que el iterador u otros procesos realizan en la tabla subyacente son visibles para el iterador. Insensible significa que si el cursor es de sólo lectura, se comportará como un cursor no sensible. Si no se trata de un cursor insensible, se comportará como un cursor sensible.

Si un iterador desplazable es estático, el tamaño de la tabla de resultados y el orden de las filas en la tabla de resultados no cambian después de abrir el iterador. Esto significa que no puede insertar datos en tablas de resultados, y si suprime una fila de una tabla de resultados, se produce un hueco por supresión. Si actualiza una fila de la tabla de resultados y como consecuencia la fila deja de ser apropiada para la tabla de resultados, se produce un hueco por actualización. Una operación de recuperación de datos realizada en un hueco produce una SQLException.

Si un iterador desplazable es dinámico, el tamaño de la tabla de resultados y el orden de las filas en la tabla de resultados pueden cambiar después de abrir el

iterador. Las filas que se insertan o suprimen con sentencias INSERT y DELETE ejecutadas por el mismo proceso de aplicación son visibles inmediatamente. Las filas que se insertan o suprimen con sentencias INSERT y DELETE ejecutadas por otros procesos de aplicación son visibles una vez confirmados los cambios.

Importante: Los servidores DB2 Database para Linux, UNIX y Windows no dan soporte a los cursores desplazables dinámicos. Puede utilizar iteradores desplazables dinámicos en las aplicaciones SQLJ sólo si dichas aplicaciones acceden a datos de servidores DB2 para z/OS en la versión 9 o la versión posterior.

Importante:

Para crear y utilizar un iterador desplazable, debe seguir estos pasos:

Procedimiento

1. Especifique una cláusula de declaración de iterador que incluya las cláusulas siguientes:
   - `implements sqlj.runtime.Scrollable`

     Esto indica que el iterador es desplazable.
   - `with (sensitivity=INSENSITIVE|SENSITIVE|ASENSITIVE)` o `with (sensitivity=SENSITIVE, dynamic=true|false)`

     `sensitivity=INSENSITIVE|SENSITIVE|ASENSITIVE` indica si las operaciones de actualización o eliminación de la tabla subyacente pueden ser visibles para el iterador. El valor por omisión del atributo "sensitivity" es `INSENSITIVE`.

     `dynamic=true|false` indica si el tamaño de la tabla de resultados o el orden de las filas en la tabla puede cambiar después de abrir el iterador. El valor por omisión del atributo "dynamic" es `false`.

   El iterador puede ser un iterador con nombre o de posición.

   Ejemplo: la siguiente cláusula de declaración de iterador declara un iterador de posición que es sensible, dinámico y desplazable:

   ```
   #sql public iterator ByPos
     implements sqlj.runtime.Scrollable
     with (sensitivity=SENSITIVE, dynamic=true)  (String);
   ```

   Ejemplo: la siguiente cláusula de declaración de iterador declara un iterador con nombre que es insensible y desplazable:

   ```
   #sql public iterator ByName
     implements sqlj.runtime.Scrollable
     with (sensitivity=INSENSITIVE) (String EmpNo);
   ```

   Restricción: No puede utilizar un iterador desplazable para seleccionar columnas de tabla de un servidor DB2 Database para Linux, UNIX y Windows cuyos tipos de datos sean los siguientes:
   - LONG VARCHAR
   - LONG VARGRAPHIC
   - BLOB
   - CLOB
   - XML
   - Un tipo diferenciado que esté basado en cualquiera de los tipos de datos anteriores de esta lista
   - Un tipo estructurado
2. Cree un objeto de iterador, que es una instancia de la clase de iterador utilizada.
3. Si desea indicar al entorno de ejecución de SQLJ la dirección inicial de la recuperación de datos, utilice el método `setFetchDirection(int` *`dirección`*`)`.

dirección puede ser `FETCH_FORWARD` o `FETCH_REVERSE`. Si no invoca setFetchDirection, la dirección de recuperación de datos es `FETCH_FORWARD`.

4. Para cada fila a la que desee acceder:

   En el caso de un iterador con nombre, siga estos pasos:

   a. Sitúe el cursor utilizando uno de los métodos listados en la tabla siguiente.

Tabla 26. Métodos sqlj.runtime.Scrollable para situar un cursor desplazable

Método	Sitúa el cursor
first[1]	En la primera fila de la tabla de resultados
last[1]	En la última fila de la tabla de resultados
previous[1,2]	En la fila anterior de la tabla de resultados
next	En la fila siguiente de la tabla de resultados
`absolute(int n)`[1,3]	Si *n*>0, en la fila *n* de la tabla de resultados. Si *n*<0 y *m* es el número de filas de la tabla de resultados, sitúa el iterador en la fila *m+n+1* de la tabla de resultados.
`relative(int n)`[1,4]	Si *n*>0, en la fila que está *n* filas después de la fila actual. Si *n*<0, en la fila que está situada *n* filas antes de la fila actual. Si *n*=0, en la fila actual.
afterLast[1]	Después de la última fila de la tabla de resultados
beforeFirst[1]	Antes de la primera fila de la tabla de resultados

Notas:

1. Este método no es aplicable a las conexiones con IBM Informix.
2. Si el cursor está situado después de la última fila de la tabla de resultados, este método posiciona el cursor en la última fila.
3. Si el valor absoluto de *n* es mayor que el número de filas de la tabla de resultados, este método posiciona el cursor después de la última fila si *n* es positivo, o antes de la primera fila si *n* es negativo.
4. Suponga que *m* es el número de filas de la tabla de resultados y *x* es el número de fila actual de la tabla de resultados. Si *n*>0 y *x+n*>*m*, el iterador se sitúa a continuación de la última fila. Si *n*<0 y *x+n*<1, el iterador se sitúa antes de la primera fila.

   b. Si necesita conocer la posición actual del cursor, utilice el método getRow, isFirst, isLast, `isBeforeFirst` o isAfterLast para obtener esa información.

      Si necesita conocer la dirección actual de recuperación de datos, invoque el método getFetchDirection.

   c. Utilice métodos accesores para recuperar la fila actual de la tabla de resultados.

   d. Si las operaciones de actualización o supresión realizadas por el iterador o por otros medios son visibles en la tabla de resultados, invoque el método getWarnings para determinar si la fila actual es un hueco.

   En el caso de un iterador de posición, siga estos pasos:

   a. Utilice una sentencia FETCH con una cláusula de dirección de la recuperación de datos para posicionar el iterador y recuperar la fila actual de la tabla de resultados. La Tabla 27 lista las cláusulas que puede utilizar para posicionar el cursor.

Tabla 27. Cláusulas de FETCH para situar un cursor desplazable

Método	Sitúa el cursor
FIRST[1]	En la primera fila de la tabla de resultados
LAST[1]	En la última fila de la tabla de resultados

Tabla 27. Cláusulas de FETCH para situar un cursor desplazable (continuación)

Método	Sitúa el cursor
PRIOR[1,2]	En la fila anterior de la tabla de resultados
NEXT	En la fila siguiente de la tabla de resultados
ABSOLUTE(*n*)[1,3]	Si *n*>0, en la fila *n* de la tabla de resultados. Si *n*<0 y *m* es el número de filas de la tabla de resultados, sitúa el iterador en la fila *m+n+1* de la tabla de resultados.
RELATIVE(*n*)[1,4]	Si *n*>0, en la fila que está *n* filas después de la fila actual. Si *n*<0, en la fila que está situada *n* filas antes de la fila actual. Si *n*=0, en la fila actual.
AFTER[1,5]	Después de la última fila de la tabla de resultados
BEFORE[1,5]	Antes de la primera fila de la tabla de resultados

Notas:

1. Este valor no se puede utilizar para conexiones con IBM Informix.
2. Si el cursor está situado después de la última fila de la tabla de resultados, este método posiciona el cursor en la última fila.
3. Si el valor absoluto de *n* es mayor que el número de filas de la tabla de resultados, este método posiciona el cursor después de la última fila si *n* es positivo, o antes de la primera fila si *n* es negativo.
4. Suponga que *m* es el número de filas de la tabla de resultados y *x* es el número de fila actual de la tabla de resultados. Si *n*>0 y *x+n>m*, el iterador se sitúa a continuación de la última fila. Si *n*<0 y *x+n*<1, el iterador se sitúa antes de la primera fila.
5. No se asignan valores a las expresiones de lenguaje principal.

   b. Si las operaciones de actualización o supresión realizadas por el iterador o por otros medios son visibles en la tabla de resultados, invoque el método getWarnings para determinar si la fila actual es un hueco.

5. Invoque el método close para cerrar el iterador.

Ejemplo

El código de programa siguiente muestra cómo utilizar un iterador con nombre para recuperar el número y apellido de un empleado de entre todas las filas de la tabla EMPLOYEE, en orden inverso. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
#sql context Ctx;             // Crear la clase de contexto de conexión Ctx
#sql  iterator ScrollIter implements sqlj.runtime.Scrollable        1
  (String EmpNo, String LastName);
{
  ...
  Ctx ctxt =
   new Ctx("jdbc:db2://sysmvs1.stl.ibm.com:5021/NEWYORK",
   userid,password,false);   // Crear objeto contexto conexión ctxt
                             // para la conexión con NEWYORK
  ScrollIter scrliter;                                              2
  #sql [ctxt]
    scrliter={SELECT EMPNO, LASTNAME FROM EMPLOYEE};
  scrliter.afterLast();
  while (scrliter.previous())                                       4a
  {
    System.out.println(scrliter.EmpNo() + " "                       4c
      + scrliter.LastName());
  }
  scrliter.close();                                                 5
}
```

Llamada a procedimientos almacenados en aplicaciones SQLJ

Para invocar un procedimiento almacenado, utilice una cláusula ejecutable que contenga una sentencia CALL de SQL.

Acerca de esta tarea

Puede ejecutar la sentencia CALL con parámetros de identificador de lenguaje principal. Puede ejecutar la sentencia CALL con parámetros literales solamente si el servidor DB2 donde se ejecuta la sentencia CALL es compatible con la ejecución dinámica de la sentencia CALL.

Estos son los pasos básicos para invocar un procedimiento almacenado:

Procedimiento

1. Asigne valores a los parámetros de entrada (IN o INOUT).
2. Invoque el procedimiento almacenado.
3. Procese los parámetros de salida (OUT o INOUT).
4. Si el procedimiento almacenado devuelve varios conjuntos de resultados, obtenga esos resultados.

Ejemplo

El código siguiente muestra la invocación de un procedimiento almacenado que tiene tres parámetros de entrada y tres parámetros de salida. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
String FirstName="TOM";            // Parámetros de entrada            1
String LastName="NARISINST";
String Address="IBM";
int CustNo;                        // Parámetros de salida
String Mark;
String MarkErrorText;
...
#sql [myConnCtx] {CALL ADD_CUSTOMER(:IN FirstName,                      2
                            :IN LastName,
                            :IN Address,
                            :OUT CustNo,
                            :OUT Mark,
                            :OUT MarkErrorText)};
                                  // Invocar el procedimiento almacenado
System.out.println("Parámetros de salida de la llamada a ADD_CUSTOMER: ");
System.out.println("Número de cliente de " + LastName + ": " + CustNo);  3
System.out.println(Mark);
If (MarkErrorText != null)
  System.out.println(" Error messages:" + MarkErrorText);
```

Figura 38. Ejemplo de invocación de un procedimiento almacenado en una aplicación SQLJ

Utilización de parámetros con nombre en las sentencias CALL en aplicaciones SQLJ

Puede utilizar parámetros con nombre para correlacionar los nombres de variable del lenguaje principal de una sentencia CALL con los nombres de parámetro en la definición del procedimiento almacenado.

Acerca de esta tarea

En el caso de los parámetros con nombre, no es necesario especificar los parámetros en la sentencia CALL en el mismo orden en el que aparecen en la definición del procedimiento almacenado. Además, no tendrá que especificar todos los parámetros en la sentencia CALL. Los parámetros sin especificar tomarán los valores por omisión que están especificados en la definición del procedimiento almacenado.

Para utilizar parámetros con nombre con sentencias CALL, realice los pasos siguientes.

Procedimiento

1. En la sentencia CALL, asigne valores a las variables del lenguaje principal IN o INOUT.

   Los parámetros con nombre apuntan a las variables del lenguaje principal. Las reglas para la denominación de los parámetros con nombre y la asignación a parámetros con nombre deben ser conformes con las reglas de los parámetros con nombre en las sentencias de SQL CALL. Puede asignar explícitamente el valor por omisión o el valor nulo a un parámetro con nombre especificando la palabra clave DEFAULT o NULL. En el caso de los parámetros para los que se especifica un valor por omisión en la sentencia CREATE PROCEDURE, puede asignar implícitamente los valores por omisión a los parámetros con nombre omitiendo dichos parámetros desde la sentencia CALL. Sólo puede saltar parámetros si todos los parámetros omitidos cuentan con valores por omisión en la definición del procedimiento almacenado.

   No puede mezclar parámetros con nombre y parámetros sin nombre en la misma sentencia CALL.
2. Procese los parámetros de salida (OUT o INOUT).
3. Si el procedimiento almacenado devuelve varios conjuntos de resultados, obtenga esos resultados.

Ejemplo

El código siguiente muestra la invocación de un procedimiento almacenado que tiene la definición siguiente:

```
CREATE PROCEDURE SALS (
  OUT retcode INTEGER,
  IN lowsal DOUBLE,
  IN medsal DOUBLE,
  IN highsal DOUBLE DEFAULT 100000,
  IN department CHAR(3) DEFAULT '---')
SPECIFIC JDBC_SALS
DYNAMIC RESULT SETS 0
DETERMINISTIC
LANGUAGE JAVA
PARAMETER STYLE JAVA
NO DBINFO
FENCED
THREADSAFE
MODIFIES SQL DATA
PROGRAM TYPE SUB
EXTERNAL NAME 'MYJAR:MyClass.sals'
```

Los parámetros de entrada de la sentencia CALL se representan mediante parámetros con nombre. La llamada al tercer parámetro y al cuarto parámetro se

realiza con los valores por omisión para el procedimiento almacenado. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
double hvLowSal=10000;              // Variables leng. principal para parám. entrada
double hvMedSal=50000;
int hvRetCode;
 // Variable del lenguaje principal para el parámetro de salida
...
#sql [myConnCtx] {CALL SALS(retcode=>:OUT hvRetCode,             1
                    lowsal=>:IN hvLowSal,
                    medsal=>:IN hvMedSal,
                    highsal=>DEFAULT)};
                                 // Llamar al procedimiento almacenado.
                                 // Utilizar implícitamente el valor
                                 // por omisión para el último parámetro
                                 // omitiéndolo.
System.out.println("Return code from SALS call: " + hvRetCode);    2
```

Recuperación de datos desde parámetros de salida de cursor en aplicaciones SQLJ

Los procedimientos almacenados de DB2 Database para Linux, UNIX y Windows pueden tener parámetros OUT del tipo cursor. Para recuperar datos de dichos parámetros en aplicaciones SQLJ, utilice iteradores u objetos ResultSet.

Acerca de esta tarea

Para recuperar datos de variables de cursor, realice los pasos siguientes.

Procedimiento

1. Defina un iterador o un objeto ResultSet para cada parámetro OUT que tiene el tipo de datos CURSOR en la definición del procedimiento almacenado.

   Se puede asignar un nombre o una posición a los iteradores para recuperar parámetros OUT de cursor.
2. Asigne valores a los parámetros de entrada.
3. Invoque el procedimiento almacenado.
4. Recupere filas de los parámetros de cursor.
   - Si declara un iterador de posición para el parámetro de cursor, utilice sentencias FETCH para recuperar los datos.
   - Si declara un iterador con nombre para el parámetro de cursor, utilice métodos NamedIterator para recuperar los datos.
   - Si define un objeto ResultSet para el parámetro de cursor, utilice los métodos ResultSet para posicionar el cursor y recuperar valores de las filas del conjunto de resultados.
5. Si el procedimiento almacenado devuelve múltiples conjuntos de resultados al abrir cursores definidos como WITH RETURN, recupere esos conjuntos de resultados.

   Un único procedimiento almacenado puede devolver datos a través de múltiples conjuntos de resultados, así como parámetros CURSOR.

Ejemplo

Un tipo de datos cursor y un procedimiento almacenado tienen las definiciones siguientes:

```
CREATE TYPE myRowType AS ROW (name VARCHAR(128))
CREATE TYPE myCursorType AS myRowType CURSOR
CREATE PROCEDURE MYPROC(IN pempNo VARCHAR(6), OUT pcv1 myCursorType)
```

```
  RESULT SETS 0
  LANGUAGE SQL
  BEGIN
     SET pcv1 = CURSOR FOR SELECT name FROM employee WHERE empNo = pempNo;
     OPEN pcv1;
  END
```

El código siguiente llama al procedimiento almacenado MYPROC y utiliza un iterador de posición para recuperar datos del cursor pcv1. Los números situados a la derecha de determinadas sentencias corresponden a pasos descritos anteriormente.

```
#sql iterator Iter (String);         // Declarar un iterador de posición
...
Iter iter = null;                    // Parámetro de salida            1
String hvPempNo="000500";            // Parámetro de entrada             2
#sql [ctx] {CALL MYPROC (:IN hvPempNo, :OUT iter)};                     3
                                     // Invocar el procedimiento almacenado
String hvEmpName = null;
while (true) {                       // Recuperar filas de conjunto de resultados
  #sql { FETCH :iter into :hvName };                                    4
  if (iter.endFetch()) break;
  System.out.println("Employee name for " + hvPempNo
    + ": " + hvEmpName);
}
```

El código siguiente llama al procedimiento almacenado MYPROC y utiliza un objeto ResultSet para recuperar datos del cursor pcv1. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
...
ResultSet rs = null;                 // Parámetro de salida              1
String hvPempNo="000500";            // Parámetro de entrada               2
#sql [ctx] {CALL MYPROC (:IN hvPempNo, :OUT rs)};                         3
                                     // Invocar el procedimiento almacenado
String hvEmpName = null;
while (rs.next()) {                  // Recuperar filas de conj. resultados 4
  hvEmpName=rs.getString(1);
  System.out.println("Employee name for " + hvPempNo
    + ": " + hvEmpName);
}
```

Recuperación de varios conjuntos de resultados de un procedimiento almacenado en una aplicación SQLJ

Algunos procedimientos almacenados devuelven uno o varios conjuntos de resultados al programa que emite la llamada mediante la inclusión de la cláusula DYNAMIC RESULT SETS *n* en la definición, con *n*>0, y mediante la apertura de cursores que están definidos con la cláusula WITH RETURN. El programa solicitante necesita recuperar el contenido de esos conjuntos de resultados.

Para recuperar las filas de esos conjuntos de resultados, siga estos pasos:

1. Adquiera un contexto de ejecución para recuperar el conjunto de resultados del procedimiento almacenado.
2. Asocie el contexto de ejecución con la sentencia CALL para el procedimiento almacenado.

   No utilice este contexto de ejecución para ninguna otra finalidad hasta que haya recuperado y procesado el último conjunto de resultados.
3. Para cada conjunto de resultados:
   a. Utilice el método getNextResultSet del ExecutionContext para recuperar el conjunto de resultados.

b. Si no conoce el contenido del conjunto de resultados, utilice el método ResultSetMetaData para obtener esta información.
c. Utilice un iterador de conjunto de resultados de SQLJ o un ResultSet de JDBC para recuperar las filas del conjunto de resultados.

Los conjuntos de resultados se devuelven al programa solicitante en el mismo orden en el que se abren los cursores del programa en el procedimiento almacenado. Cuando no existen más conjuntos de resultados para recuperar, getNextResultSet devuelve un valor nulo.

Existen dos formatos de getNextResultSet:

```
getNextResultSet();
getNextResultSet(int actual);
```

Cuando invoca el primer formato de getNextResultSet, SQLJ cierra el conjunto de resultados que está abierto actualmente y pasa al conjunto de resultados siguiente. Cuando invoca el segundo formato de getNextResultSet, el valor de *actual* indica lo que SQLJ realiza con el conjunto de resultados abierto actualmente antes de pasar al conjunto de resultados siguiente:

java.sql.Statement.CLOSE_CURRENT_RESULT
Especifica que el objeto ResultSet actual se cierra cuando se devuelve el objeto ResultSet siguiente.

java.sql.Statement.KEEP_CURRENT_RESULT
Especifica que el objeto ResultSet actual permanece abierto cuando se devuelve el objeto ResultSet siguiente.

java.sql.Statement.CLOSE_ALL_RESULTS
Especifica que todos los objetos ResultSet abiertos se cierran cuando se devuelve el objeto ResultSet siguiente.

El código siguiente invoca un procedimiento almacenado que devuelve varios conjuntos de resultados. En este ejemplo se supone que el peticionario no conoce el número de conjuntos de resultados que se deben devolver ni el contenido de ellos. También se supone que el valor de autoCommit es false. Los números situados a la derecha de determinadas sentencias corresponden a pasos descritos anteriormente.

```
ExecutionContext execCtx=myConnCtx.getExecutionContext();      1
#sql [myConnCtx, execCtx] {CALL MULTRSSP()};                   2
            // MULTRSSP devuelve varios conjuntos de resultados
ResultSet rs;
while ((rs = execCtx.getNextResultSet()) != null)              3a
{
  ResultSetMetaData rsmeta=rs.getMetaData();                   3b
  int numcols=rsmeta.getColumnCount();
  while (rs.next())                                            3c
  {
    for (int i=1; i<=numcols; i++)
    {
      String colval=rs.getString(i);
      System.out.println("Columna " + i + "valor es " + colval);
    }
  }
}
```

Figura 39. Recuperación de conjuntos de resultados de un procedimiento almacenado

Objetos LOB en aplicaciones SQLJ con el controlador IBM Data Server Driver para JDBC y SQLJ

Con IBM Data Server Driver para JDBC y SQLJ, puede insertar datos LOB en expresiones `Clob` o `Blob` de lenguaje principal o actualizar columnas CLOB, BLOB o DBCLOB a partir de expresiones `Clob` o `Blob` de lenguaje principal. Puede también declarar iteradores con tipos de datos `Clob` o `Blob` para recuperar datos de columnas CLOB, BLOB o DBCLOB.

Recuperación o actualización de datos LOB: para recuperar datos de una columna BLOB, declare un iterador que incluya el tipo de datos `Blob` o `byte[]`. Para recuperar datos de una columna CLOB o DBCLOB, declare un iterador en el que la correspondiente columna tenga un tipo de datos `Clob`.

Para actualizar datos de una columna BLOB, utilice una expresión de lenguaje principal cuyo tipo de datos sea `Blob`. Para actualizar datos de una columna CLOB o DBCLOB, utilice una expresión de lenguaje principal cuyo tipo de datos sea `Clob`.

Modalidad continua progresiva o localizadores de LOB: En las aplicaciones SQLJ, puede utilizar la modalidad continua progresiva, también conocida como formato de datos dinámico, o localizadores de LOB de la misma manera que los utiliza en las aplicaciones JDBC.

Tipos de datos Java para recuperar o actualizar datos de columnas LOB en aplicaciones SQLJ

Cuando la propiedad deferPrepares tiene el valor "true", y el controlador IBM Data Server Driver para JDBC y SQLJ procesa una sentencia de SQLJ no personalizada que incluye expresiones de lenguaje principal, el controlador puede necesitar realizar tareas adicionales de proceso para determinar los tipos de datos. Este proceso adicional puede afectar al rendimiento.

Cuando el controlador JDBC no puede determinar inmediatamente el tipo de datos de un parámetro que se utiliza con una columna LOB, es necesario elegir un tipo de datos para el parámetro que sea compatible con el tipo de datos LOB.

Parámetros de entrada para columnas BLOB

Para los parámetros de entrada de columnas BLOB, puede utilizar cualquiera de las dos técnicas siguientes:

- Utilice una variable de entrada java.sql.Blob, que se corresponde exactamente con una columna BLOB:

  ```
  java.sql.Blob blobData;
  #sql {CALL STORPROC(:IN blobData)};
  ```

  Para poder utilizar una variable de entrada java.sql.Blob, es necesario crear un objeto java.sql.Blob y luego llenar con datos ese objeto.
- Utilice un parámetro de entrada de tipo sqlj.runtime.BinaryStream. Un objeto sqlj.runtime.BinaryStream es compatible con un tipo de datos BLOB. Por ejemplo:

  ```
  java.io.ByteArrayInputStream byteStream =
    new java.io.ByteArrayInputStream(byteData);
  int numBytes = byteData.length;
  sqlj.runtime.BinaryStream binStream =
    new sqlj.runtime.BinaryStream(byteStream, numBytes);
  #sql {CALL STORPROC(:IN binStream)};
  ```

No puede utilizar esta técnica para parámetros INOUT.

Parámetros de salida para columnas BLOB

Para los parámetros de salida INOUT o de salida de columnas BLOB, puede utilizar la técnica siguiente:

- Declare el parámetro de salida o variable INOUT con un tipo de datos java.sql.Blob:

```
java.sql.Blob blobData = null;
#sql CALL STORPROC (:OUT blobData)};
```

```
java.sql.Blob blobData = null;
#sql CALL STORPROC (:INOUT blobData)};
```

Parámetros de entrada para columnas CLOB

Para los parámetros de entrada de columnas CLOB, puede utilizar una de las técnicas siguientes:

- Utilice una variable de entrada java.sql.Clob, que se corresponde exactamente con una columna CLOB:

```
#sql CALL STORPROC(:IN clobData)};
```

  Para poder utilizar una variable de entrada java.sql.Clob, es necesario crear un objeto java.sql.Clob y luego llenar con datos ese objeto.
- Utilice uno de los tipos siguientes de parámetros IN de secuencia:
  - Un parámetro de entrada sqlj.runtime.CharacterStream:

```
java.lang.String charData;
java.io.StringReader reader = new java.io.StringReader(charData);
sqlj.runtime.CharacterStream charStream =
  new sqlj.runtime.CharacterStream (reader, charData.length);
#sql {CALL STORPROC(:IN charStream)};
```

  - Un parámetro sqlj.runtime.UnicodeStream, para datos Unicode UTF-16:

```
byte[] charDataBytes = charData.getBytes("UnicodeBigUnmarked");
java.io.ByteArrayInputStream byteStream =
  new java.io.ByteArrayInputStream(charDataBytes);
sqlj.runtime.UnicodeStream uniStream =
  new sqlj.runtime.UnicodeStream(byteStream, charDataBytes.length );
#sql {CALL STORPROC(:IN uniStream)};
```

  - Un parámetro sqlj.runtime.AsciiStream, para datos ASCII:

```
byte[] charDataBytes = charData.getBytes("US-ASCII");
java.io.ByteArrayInputStream byteStream =
  new java.io.ByteArrayInputStream (charDataBytes);
sqlj.runtime.AsciiStream asciiStream =
  new sqlj.runtime.AsciiStream (byteStream, charDataBytes.length);
#sql {CALL STORPROC(:IN asciiStream)};
```

  Para estas llamadas es necesario especificar la longitud exacta de los datos de entrada. No puede utilizar esta técnica para parámetros INOUT.
- Utilice un parámetro de entrada java.lang.String:

```
java.lang.String charData;
#sql {CALL STORPROC(:IN charData)};
```

Parámetros de salida para columnas CLOB

Para los parámetros de salida o INOUT de columnas CLOB, puede utilizar una de las técnicas siguientes:

- Utilice una variable de salida java.sql.Clob, que se corresponde exactamente con una columna CLOB:

```
java.sql.Clob clobData = null;
#sql CALL STORPROC(:OUT clobData)};
```

- Utilice una variable de salida java.lang.String:

```
java.lang.String charData = null;
#sql CALL STORPROC(:OUT charData)};
```

  Esta técnica se debe utilizar solamente si sabe que la longitud de los datos recuperados es menor o igual que 32KB. En otro caso, los datos se truncan.

Parámetros de salida para columnas DBCLOB

Los parámetros de salida o INOUT de DBCLOB para procedimientos almacenados no están soportados.

SQLJ y JDBC en la misma aplicación

Puede combinar cláusulas SQLJ y llamadas JDBC en un mismo programa.

Para ello, debe seguir estos pasos:

- Utilice una Connection JDBC para crear ConnectionContext SQLJ u obtenga una Connection JDBC a partir de un ConnectionContext SQLJ.
- Utilice un iterador SQLJ para obtener datos a partir de un ResultSet JDBC o genere un ResultSet JDBC a partir de un iterador SQLJ.

Creación de un contexto de conexión SQLJ a partir de una conexión JDBC - Siga estos pasos:

1. Ejecute una cláusula de declaración de conexión SQLJ para crear una clase ConnectionContext.
2. Cargue el controlador u obtenga una instancia de DataSource.
3. Invoque el método DriverManager.getConnection o DataSource.getConnection de SQLJ para obtener un objeto Connection JDBC.
4. Invoque el constructor ConnectionContext con el objeto Connection como argumento para crear el objeto ConnectionContext.

Obtención de una conexión JDBC a partir de un contexto de conexión SQLJ - Siga estos pasos:

1. Ejecute una cláusula de declaración de conexión SQLJ para crear una clase ConnectionContext.
2. Cargue el controlador u obtenga una instancia de DataSource.
3. Invoque el constructor ConnectionContext con el URL del controlador y cualquier otro parámetro necesario como argumentos para crear el objeto ConnectionContext.
4. Invoque el método ConnectionContext.getConnection de JDBC para crear el objeto Connection de JDBC.

Consulte "Conexión con una fuente de datos utilizando SQLJ" para obtener más información sobre las conexiones SQLJ.

Obtención de conjuntos de resultados JDBC utilizando iteradores SQLJ: Utilice la *sentencia de conversión a iterador* para manejar un conjunto de resultados JDBC como un iterador SQLJ. Este es el formato general de una sentencia de conversión a iterador:

```
#sql iterador={CAST :conjunto-resultados};
```

Para convertir satisfactoriamente un conjunto de resultados en un iterador, éste debe cumplir las reglas siguientes:

- El iterador debe estar declarado como público.
- Si el iterador es un iterador de posición, el número de columnas del conjunto de resultados debe coincidir con el número de columnas del iterador. Además, el tipo de datos de cada columna del conjunto de resultados debe coincidir con el tipo de datos de la columna correspondiente del iterador.
- Si el iterador es un iterador de nombre, el nombre de cada método accesor debe coincidir con el nombre de una columna del conjunto de resultados. Además, el tipo de datos del objeto devuelto por un método accesor debe coincidir con el tipo de datos de la columna correspondiente del conjunto de resultados.

El código mostrado en la Figura 40 crea y ejecuta una consulta utilizando una llamada JDBC, ejecuta una sentencia de conversión a iterador para convertir el conjunto de resultados JDBC en un iterador SQLJ y obtiene filas del conjunto de resultados utilizando el iterador.

```
#sql public iterator ByName(String LastName, Date HireDate);      1
public void HireDates(ConnectionContext connCtx, String whereClause)
{

  ByName nameiter;       // Declarar objeto de la clase ByName
  Connection conn=connCtx.getConnection();
                           // Crear la conexión JDBC
  Statement stmt = conn.createStatement();                         2
  String query = "SELECT LASTNAME, HIREDATE FROM EMPLOYEE";
  query+=whereClause;   // Crear la consulta
  ResultSet rs = stmt.executeQuery(query);                         3
  #sql [connCtx] nameiter = {CAST :rs};                            4
  while (nameiter.next())
  {
    System.out.println( nameiter.LastName() + " se contrató el "
      + nameiter.HireDate());

  }
  nameiter.close();                                                5
  stmt.close();
}
```

Figura 40. Conversión de un conjunto de resultados JDBC en un iterador SQLJ

Notas para la Figura 40:

Nota	Descripción
1	Esta cláusula de SQLJ crea la clase de iterador de nombre ByName, cuyos métodos accesores LastName() y HireDate() obtienen datos de las columnas LASTNAME y HIREDATE de la tabla de resultados.
2	Esta sentencia y las dos sentencias que le siguen crean y preparan una consulta para su ejecución dinámica mediante JDBC.
3	Esta sentencia de JDBC ejecuta la sentencia SELECT y asigna la tabla de resultados al conjunto de resultados rs.
4	Esta cláusula de conversión a iterador convierte el ResultSet rs de JDBC en el iterador nameiter de SQLJ y las sentencias que siguen utilizan nameiter para obtener valores de la tabla de resultados.
5	El método nameiter.close() cierra el iterador de SQLJ y el ResultSet rs de JDBC.

Generación de conjuntos de resultados de JDBC a partir de iteradores de SQLJ: utilice el método getResultSet para generar un ResultSet a partir de un iterador de SQLJ. Cada iterador de SQLJ tiene un método getResultSet. Después de acceder al ResultSet en el que se basa un iterador, necesita recuperar filas utilizando solamente ResultSet.

El código mostrado en la Figura 41 genera un iterador de posición para una consulta, convierte el iterador en un conjunto de resultados y utiliza métodos de JDBC para recuperar filas de la tabla.

```
#sql iterator EmpIter(String, java.sql.Date);
{
...
  EmpIter iter=null;
  #sql [connCtx] iter=
    {SELECT LASTNAME, HIREDATE FROM EMPLOYEE};                  1
  ResultSet rs=iter.getResultSet();                             2
  while (rs.next())                                             3
  { System.out.println(rs.getString(1) + " was hired in " +
      rs.getDate(2));
  }
  rs.close();                                                   4
}
```

Figura 41. Conversión de un iterador SQLJ en un conjunto de resultados JDBC

Notas para la Figura 41:

Nota	Descripción
1	Esta cláusula de SQLJ ejecuta la sentencia SELECT, construye un objeto iterador que contiene la tabla de resultados correspondiente a la sentencia SELECT y asigna el objeto iterador a la variable iter.
2	El método getResultSet() accede al ResultSet en el que se basa el propio iterador.
3	Los métodos getString() y getDate() de JDBC obtienen valores a partir del ResultSet. El método next() coloca el cursor en la fila siguiente del ResultSet.
4	El método rs.close() cierra el iterador de SQLJ y el ResultSet.

Reglas y restricciones para utilizar conjuntos de resultados de JDBC en aplicaciones SQLJ: tenga en cuenta las reglas y restricciones siguientes al escribir aplicaciones de SQLJ que incluyen conjuntos de resultados de JDBC:

- No puede convertir un ResultSet en un iterador de SQLJ si el ResultSet y el iterador tienen valores diferentes para el atributo de capacidad de retención del cursor.

  Un ResultSet de JDBC o iterador de SQLJ pueden permanecer abiertos después de una operación COMMIT. Para un ResultSet de JDBC, esta característica se controla mediante la propiedad `resultSetHoldability` de IBM Data Server Driver para JDBC y SQLJ. Para un iterador de SQLJ, esta característica está controlada por el parámetro `with holdability` de la sentencia de declaración del iterador. No está soportada la conversión de un ResultSet con capacidad de retención en un iterador de SQLJ que no tenga esa capacidad, ni la conversión de un ResultSet sin capacidad de retención en un iterador que sí la tenga.
- Cierre el iterador o el objeto ResultSet subyacente en cuanto el programa deje de utilizar el iterador o ResultSet, y antes de que finalice el programa.

  Al cerrar el iterador se cierra también el objeto ResultSet. Al cerrar el objeto ResultSet se cierra también el objeto iterador. En general, es mejor cerrar el objeto que se ha utilizado en último lugar.

- Para IBM Data Server Driver para JDBC y SQLJ, que da soporte a iteradores desplazables y a objetos ResultSet desplazables y actualizables, son aplicables las restricciones siguientes:
  - Los iteradores desplazables tienen las mismas restricciones que sus objetos ResultSet de JDBC subyacentes.
  - No puede convertir un ResultSet de JDBC que no sea actualizable en un iterador SQLJ que sea actualizable.

Control de la ejecución de sentencias de SQL en SQLJ

Puede utilizar determinados métodos de la clase ExecutionContext de SQLJ para controlar o supervisar la ejecución de sentencias de SQL.

Acerca de esta tarea

Siga estos pasos para utilizar métodos de ExecutionContext:

Procedimiento

1. Adquiera el contexto de ejecución por omisión a partir del contexto de conexión.

   Existen dos formas de adquirir un contexto de ejecución:

   - Adquiera el contexto de ejecución por omisión a partir del contexto de conexión. Por ejemplo:

     ```
     ExecutionContext execCtx = connCtx.getExecutionContext();
     ```

   - Cree un nuevo contexto de ejecución invocando el constructor de ExecutionContext. Por ejemplo:

     ```
     ExecutionContext execCtx=new ExecutionContext();
     ```

2. Asocie el contexto de ejecución a una sentencia de SQL.

   Para ello, especifique un contexto de ejecución a continuación del contexto de conexión en la cláusula de ejecución donde reside la sentencia de SQL.

3. Invoque métodos de ExecutionContext.

   Algunos métodos de ExecutionContext son aplicables antes de ejecutar la sentencia de SQL asociada, mientras que otros son aplicables solo después de ejecutar su sentencia de SQL asociada.

   Por ejemplo, puede utilizar el método getUpdateCount para contar el número de filas que son suprimidas por una sentencia DELETE después de ejecutar esa sentencia.

Ejemplo

El código de programa siguiente muestra cómo adquirir un contexto de ejecución y luego utilizar el método getUpdateCount en ese contexto de ejecución para determinar el número de filas que fueron suprimidas por una sentencia DELETE. Los números situados a la derecha de determinadas sentencias corresponden a pasos descritos anteriormente.

```
ExecutionContext execCtx=new ExecutionContext();                          1
#sql [connCtx, execCtx] {DELETE FROM EMPLOYEE WHERE SALARY > 10000};      2
System.out.println("Suprimidas " + execCtx.getUpdateCount() + " filas"); 3
```

Identificadores de fila (ROWID) en SQLJ con el controlador IBM Data Server Driver para JDBC y SQLJ

DB2 para z/OS y DB2 para i dan soporte al tipo de datos ROWID para una columna en una tabla. Un ROWID es un valor que identifica una fila de una tabla de una forma exclusiva.

Aunque IBM Informix también es compatible con la utilización de identificadores de fila, el tipo de datos de estos identificadores es INTEGER. Puede seleccionar una columna de identificador de fila de IBM Informix y asignarla a una variable cuyo tipo de datos es un entero de cuatro bytes.

Si utiliza columnas cuyo tipo de datos es ROWID en programas SQLJ, es necesario que personalice esos programas.

JDBC 4.0 incluye la interfaz java.sql.RowId, que puede ser utilizada en iteradores y parámetros de la sentencia CALL. Si no tiene instalado JDBC 4.0, puede utilizar la clase com.ibm.db2.jcc.DB2RowID, que es específica de IBM Data Server Driver para JDBC y SQLJ. Para un iterador, puede también utilizar el tipo de objeto byte[] para recuperar valores de ROWID.

El código de programa siguiente muestra un ejemplo de un iterador que se utiliza para seleccionar valores de una columna ROWID:

```
#sql iterator PosIter(int,String,java.sql.RowId);
                                // Declarar un iterador de posición para
                    // recuperar valores ITEM_ID (INTEGER),
                    // ITEM_FORMAT (VARCHAR) y ITEM_ROWID (ROWID)
                    // de la tabla ROWIDTAB
{
  PosIter positrowid;       // Declarar objeto de la clase PosIter
  java.sql.RowId rowid = null;
  int id = 0;
  String i_fmt = null;
                                // Declarar expresiones de lenguaje principal
  #sql [ctxt] positrowid =
    {SELECT ITEM_ID, ITEM_FORMAT, ITEM_ROWID FROM ROWIDTAB
       WHERE ITEM_ID=3};
                                // Asignar tabla de resultados de SELECT
                                // al objeto iterador positrowid
  #sql {FETCH :positrowid INTO :id, :i_fmt, :rowid};
                                // Recuperar primera fila
  while (!positrowid.endFetch())
                                // Determinar si FETCH devolvió una fila
    {System.out.println("Item ID " + id + " Item format " +
       i_fmt + " Item ROWID ");
       MyUtilities.printBytes(rowid.getBytes());
                           // Usar el método getBytes para convertir
                           // el valor a bytes para la impresión.
                           // Llamar a un método definido por el usuario denominado
                           // printBytes (no se muestra) para imprimir
                           // el valor.
       #sql {FETCH :positrowid INTO :id, :i_fmt, :rowid};
                                // Recuperar la fila siguiente
    }
  positrowid.close();           // Cerrar el iterador
}
```

Figura 42. Ejemplo de utilización de un iterador para recuperar valores de ROWID

El código de programa siguiente muestra un ejemplo de invocación de un procedimiento almacenado que acepta tres parámetros ROWID: un parámetro IN,

un parámetro OUT y un parámetro INOUT.

```
java.sql.RowId in_rowid = rowid;
java.sqlRowId out_rowid = null;
java.sql.RowId inout_rowid = rowid;
                                  // Declarar un parámetro IN, OUT e
                                  // INOUT ROWID
...
#sql [myConnCtx] {CALL SP_ROWID(:IN in_rowid,
                             :OUT out_rowid,
                             :INOUT inout_rowid)};
                                  // Invocar el procedimiento almacenado
System.out.println("Valores de parámetros de llamada a SP_ROWID: ");
System.out.println("Valor de parámetro OUT ");
MyUtilities.printBytes(out_rowid.getBytes());
                           // Usar el método getBytes para convertir
                           // convertir el valor en bytes para imprimirlo
                           // Llamar a un método definido por el usuario denominado
                           // printBytes (no se muestra) para imprimir
                           // el valor.
System.out.println("Valor de parámetro INOUT");
MyUtilities.printBytes(inout_rowid.getBytes());
```

Figura 43. Ejemplo de invocación de un procedimiento almacenado con un parámetro ROWID

Valores de TIMESTAMP WITH TIME ZONE en aplicaciones SQLJ

DB2 para z/OS da soporte a las columnas de tabla con el tipo de datos TIMESTAMP WITH TIME ZONE. IBM Data Server Driver para JDBC y SQLJ da soporte a la actualización en una columna con el tipo de datos TIMESTAMP WITH TIME ZONE y a la recuperación a partir de ésta en los programas SQLJ.

Al actualizar o recuperar un valor TIMESTAMP WITH TIME ZONE, o llamar a un procedimiento almacenado con un parámetro TIMESTAMP WITH TIME ZONE, tiene que utilizar variables de lenguaje principal que sean objetos com.ibm.db2.jcc.DBTimestamp para conservar la información de huso horario. Si utiliza objetos java.sql.Timestamp para pasar valores TIMESTAMP WITH TIME ZONE a y desde el servidor de datos, perderá la información de huso horario.

Dado que la clase com.ibm.db2.jcc.DBTimestamp es una clase exclusiva de IBM Data Server Driver para JDBC y SQLJ, si ejecuta una aplicación SQLJ no personalizada que utiliza objetos com.ibm.db2.jcc.DBTimestamp, la aplicación recibe una SQLException.

Ejemplos

Supongamos que la tabla TSTABLE tiene una sola columna, TSCOL, que tiene el tipo de datos TIMESTAMP WITH TIME ZONE. El código siguiente asigna un valor de indicación de fecha y hora a una columna y recupera el valor de la columna.

```
#sql iterator TSIter(com.ibm.db2.jcc.DBTimestamp TSVar);
{
  ...
  java.util.TimeZone esttz = java.util.TimeZone.getTimeZone("EST");
                             // Establecer el huso horario en UTC-5
  java.util.Calendar estcal= java.util.Calendar.getInstance(esttz);
                             // Crear una instancia de calendario
                             //  con el huso horario EST
  java.sql.Timestamp ts =
   java.sql.Timestamp.valueOf("2009-02-27 21:22:33.444444");
                             // Inicializar un objeto de indicación de fecha y hora
```

```
                                //  con el valor de fecha y hora que se
                                //  quiere poner en la tabla
  com.ibm.db2.jcc.DBTimestamp dbts =
    new com.ibm.db2.jcc.DBTimestamp(ts,estcal);
                                // Crear un objeto de fecha y hora que
                                //  incluya el huso horario
  #sql[ctx] {INSERT INTO TSTABLE (TSCOL) VALUES (:dbts)};
                                // Insertar el objeto de fecha y hora en
                                //  la tabla
  #sql[ctx] {COMMIT};

  TSIter iter = null;
  #sql [ctx] iter = {SELECT TSCOL FROM TSTABLE};
                                // Asignar tabla de resultados de SELECT
  while (iter.next()) {
    System.out.println ("Timestamp = " +
      ((com.ibm.db2.jcc.DBTimestamp)iter.TSVar()).toDBString(true));
                                // Usar el método accesor TSVar para recuperar
                                //  el valor TIMESTAMP WITH TIME ZONE
                                //  y convertirlo a un valor DBTimestamp,
                                //  y recuperar su representación de serie.
                                //  Valor recuperado:
                                //  2009-02-27 21:22:33.444444-05:00
  }
}
```

Supongamos que el procedimiento almacenado TSSP tiene un solo parámetro INOUT, TSPARM, que tiene el tipo de datos TIMESTAMP WITH TIME ZONE. El código siguiente llama al procedimiento almacenado con un valor de indicación de fecha y hora que incluye un huso horario, y recupera un valor de parámetro con un valor de indicación de fecha y hora que incluye un huso horario.

```
{
  ...
  java.util.TimeZone esttz = java.util.TimeZone.getTimeZone("EST");
                                // Establecer el huso horario en UTC-5
  java.util.Calendar estcal= java.util.Calendar.getInstance(esttz);
                                // Crear una instancia de calendario
                                //  con el huso horario EST
  java.sql.Timestamp ts =
   java.sql.Timestamp.valueOf("2009-02-27 21:22:33.444444");
                                // Inicializar un objeto de indicación de fecha y hora
                                //  con el valor de indicación de fecha y hora que se
                                //  quiere pasar al procedimiento almacenado
  com.ibm.db2.jcc.DBTimestamp dbts =
    new com.ibm.db2.jcc.DBTimestamp(ts,estcal);
                                // Crear un objeto de indicación de fecha y hora que
                                //  incluye el huso horario que se
                                //  pasará al procedimiento almacenado
  #sql[ctx] { CALL TSSP (:INOUT dbts) };
  System.out.println ("Output parameter: " + dbts.toDBString (true));
                                //  Llamar al procedimiento almacenado con
                                //  el valor de indicación de fecha y hora como entrada
                                //  y recuperar un valor de indicación de fecha y hora
                                //  con un huso horario en el mismo
                                //  parámetro
}
```

Tipos diferenciados en aplicaciones SQLJ

En un programa SQLJ, puede crear un tipo diferenciado utilizando la sentencia CREATE DISTINCT TYPE en una cláusula ejecutable.

Puede también utilizar CREATE TABLE en una cláusula ejecutable para crear una tabla que incluya una columna de ese tipo. Cuando se recuperan datos de una columna del tipo mencionado, o se actualiza una columna de dicho tipo, se utilizan expresiones o variables del lenguaje principal Java con tipos de datos que corresponden a los tipos incorporados en que se basan los tipos diferenciados.

El ejemplo siguiente crea un tipo diferenciado que está basado en un tipo INTEGER, crea una tabla con una columna de ese tipo, inserta una fila en la tabla

y recupera la fila de la tabla:

```
String empNumVar;
int shoeSizeVar;
...
#sql [myConnCtx] {CREATE DISTINCT TYPE SHOESIZE AS INTEGER WITH COMPARISONS};
                                                  // Crear tipo diferenciado
#sql [myConnCtx] {COMMIT}; // Confirmar la creación
#sql [myConnCtx] {CREATE TABLE EMP_SHOE
  (EMPNO CHAR(6), EMP_SHOE_SIZE SHOESIZE)};
                                                  // Crear tabla usando tipo diferenciado
#sql [myConnCtx] {COMMIT}; // Confirmar la creación
#sql [myConnCtx] {INSERT INTO EMP_SHOE
  VALUES('000010',6)};     // Insertar una fila en la tabla
#sql [myConnCtx] {COMMIT}; // Confirmar la inserción
#sql [myConnCtx] {SELECT EMPNO, EMP_SHOE_SIZE
  INTO :empNumVar, :shoeSizeVar
  FROM EMP_SHOE};          // Recuperar la fila
System.out.println("Employee number: " + empNumVar +
  " Shoe size: " + shoeSizeVar);
```

Figura 44. Definición y utilización de un tipo diferenciado

Invocación de procedimientos almacenados con parámetros ARRAY en aplicaciones SQLJ

Las aplicaciones SQLJ que se ejecutan bajo IBM Data Server Driver para JDBC y SQLJ y que conectan con fuentes de datos DB2 Database para Linux, UNIX y Windows pueden invocar procedimientos almacenados que tienen parámetros ARRAY.

Puede utilizar objetos java.sql.Array como parámetros IN, OUT o INOUT en un procedimiento almacenado.

Para los parámetros IN o INOUT, utilice el método DB2Connection.createArrayOf (JDBC 3.0 o anterior) o el método Connection.createArrayOf (JDBC 4.0 o posterior) para crear un objeto java.sql.Array.

Existen dos formas de recuperar datos de un parámetro de salida ARRAY de un procedimiento almacenado:

- Utilice el método java.sql.Array.getArray para recuperar el contenido de un parámetro de salida y colocarlo en una matriz Java.
- Utilice un método java.sql.Array.getResultSet para recuperar los datos del parámetro de salida y colocarlos en un objeto ResultSet. Luego utilice métodos ResultSet para recuperar elementos de la matriz. Cada fila de ResultSet contiene dos columnas:
  - Un índice en la matriz, que comienza en 1
  - El elemento de matriz

  Necesita recuperar los elementos de la matriz a partir del ResultSet utilizando el método getObject.

Ejemplo: suponga que los parámetros de entrada y salida IN_PHONE y OUT_PHONE del procedimiento almacenado GET_EMP_DATA son matrices que están definidas de esta manera:

```
CREATE TYPE PHONENUMBERS AS VARCHAR(10) ARRAY[5]
```

Invoque GET_EMP_DATA con los dos parámetros.

```
Connection con;
String type = "CHAR";
String [] contents = {"1234", "5678", "9101"};
...
```

```
com.ibm.db2.jcc.DB2Connection db2con = (com.ibm.db2.jcc.DB2Connection) con;
                                    // Convierta la conexión como DB2Connection
                                    // para poder utilizar el método
                                    // DB2Connection.createArrayOf
java.sql.Array inPhoneData = db2con.createArrayOf(type, contents);
java.sql.Array outPhoneData;
try {
  #sql [db2con] {CALL GET_EMP_DATA(:IN inPhoneData, :OUT outPhoneData ) };
}
catch (SQLException e)
{
  throw e;
}
ResultSet rs = outPhoneData.getResultSet();
while (rs.next()) {
  String phoneNum = (String)rs.getObject(2); // Obtener número de teléfono
  System.out.println("Número de teléfono = " + phoneNum);
}
```

Puntos de salvaguarda en aplicaciones SQLJ

Cuando se utiliza IBM Data Server Driver para JDBC y SQLJ, puede incluir en su programa SQLJ cualquier formato de la sentencia SAVEPOINT de SQL.

Un punto de salvaguarda de SQL representa el estado que tienen datos y esquemas en un momento determinado dentro de una unidad de trabajo. Existen sentencias de SQL para establecer un punto de salvaguarda, liberar un punto de salvaguarda y para restaurar datos y esquemas al estado representado por el punto de salvaguarda.

El ejemplo siguiente muestra cómo establecer un punto de salvaguarda, retrotraer trabajos hasta un punto de salvaguarda y liberar el punto de salvaguarda.

Figura 45. Establecimiento, retrotracción y liberación de un punto de salvaguarda en una aplicación SQLJ

```
#sql context Ctx;             // Crear la clase de contexto de conexión Ctx
String empNumVar;
int shoeSizeVar;
...
try {                       // Cargar el controlador JDBC
  Class.forName("com.ibm.db2.jcc.DB2Driver");
}
catch (ClassNotFoundException e) {
   e.printStackTrace();
}
Connection jdbccon=
  DriverManager.getConnection("jdbc:db2://sysmvs1.stl.ibm.com:5021/NEWYORK",
    userid,password);
                              // Crear objeto de conexión jdbccon de JDBC
jdbccon.setAutoCommit(false); // No realizar confirmación automática
Ctx ctxt=new Ctx(jdbccon);
                              // Crear objeto de contexto de conexión myConnCtx
                              // para la conexión con NEWYORK
...                             // Ejecutar SQL
#sql [ctxt] {COMMIT};         // Confirmar la transacción
                              // Confirmar la creación
#sql [ctxt]
  {INSERT INTO EMP_SHOE VALUES ('000010', 6)};
                              // Insertar una fila
#sql [ctxt]
  {SAVEPOINT SVPT1 ON ROLLBACK RETAIN CURSORS};
                              // Crear un punto de salvaguarda
...
```

```
#sql [ctxt]
  {INSERT INTO EMP_SHOE VALUES ('000020', 10)};
                                // Insertar otra fila
#sql [ctxt] {ROLLBACK TO SAVEPOINT SVPT1};
                                // Retrotraer trabajo hasta el punto
                                // después de la primera inserción
...
#sql [ctxt] {RELEASE SAVEPOINT SVPT1};
                                // Liberar el punto de salvaguarda
ctx.close();                   // Cerrar el contexto de conexión
```

Datos XML en aplicaciones SQLJ

En las aplicaciones SQLJ, puede almacenar en columnas XML y recuperar datos de columnas XML.

En tablas DB2, se utiliza el tipo de datos incorporado XML para almacenar datos XML en una columna en forma de conjunto estructurado de nodos en formato de árbol.

Las aplicaciones SQLJ pueden enviar datos XML al servidor de datos o recuperar datos XML del servidor de datos de una de estas maneras:

- Como datos XML textuales
- Como datos XML binarios (datos que se encuentran en el formato Extensible Dynamic Binary XML DB2 Client/Server Binary XML), si el servidor de datos los admite

En las aplicaciones SQLJ, se puede realizar lo siguiente:

- Almacenar un documento XML completo en una columna XML mediante sentencias INSERT, UPDATE o MERGE.
- Recuperar un documento XML completo de una columna XML mediante iteradores o sentencias SELECT de una sola fila.
- Recuperar una secuencia de un documento en una columna XML mediante la función XMLQUERY de SQL para recuperar la secuencia en la base de datos y, a continuación, utilizar iteradores o sentencias SELECT de una sola fila para recuperar los datos de la serie XML serializada en una variable de aplicación.
- Recuperar una secuencia de un documento en una columna XML mediante una expresión XQuery, a la que se añade la serie 'XQUERY' como prefijo, para recuperar los elementos de la secuencia en una tabla de resultados de la base de datos. En dicha base de datos, cada fila de la tabla de resultados representa un elemento de la secuencia. A continuación, se utilizan iteradores o sentencias SELECT de una sola fila para recuperar los datos en variables de aplicación.
- Recuperar una secuencia de un documento en una columna XML en forma de una tabla definida por el usuario mediante la función XMLTABLE de SQL para definir la tabla de resultados y recuperarla. A continuación, se utilizan iteradores o sentencias SELECT de una sola fila para recuperar los datos de la tabla de resultados en variables de la aplicación.
- Puede actualizar o recuperar datos XML como datos XML textuales. Como alternativa, para las conexiones con un servidor de datos que dé soporte a los datos XML binarios, puede actualizar o recuperar datos XML como datos XML binarios.

  Para la recuperación de datos, puede utilizar la propiedad xmlFormat Datasource o Connection para controlar si el formato de los datos recuperados es XML de texto o XML binario.

Para la actualización de datos en las columnas XML, xmlFormat no tiene ningún efecto. Si los datos de entrada son datos XML binarios, y el servidor de datos no da soporte a los datos XML binarios, los datos de entrada se convierten en datos XML de texto. De otro modo, no tiene lugar ninguna conversión.

El formato de datos XML es transparente para la aplicación. El almacenamiento y la recuperación de datos XML binarios en un servidor de datos DB2 para z/OS necesita la versión 4.9 o posterior de IBM Data Server Driver para JDBC y SQLJ. El almacenamiento y la recuperación de datos XML binarios en un servidor de datos DB2 Database para Linux, UNIX y Windows necesita la versión 4.11 o posterior de IBM Data Server Driver para JDBC y SQLJ.

Se pueden utilizar objetos `java.sql.SQLXML` de JDBC 4.0 para recuperar y actualizar datos de columnas XML. Las invocaciones de métodos de metadatos, tales como ResultSetMetaData.getColumnType, devuelven el valor entero `java.sql.Types.SQLXML` para una columna de tipo XML.

Actualizaciones de columnas XML en aplicaciones de SQLJ

En una aplicación SQLJ, se pueden actualizar o insertar datos en columnas XML de una tabla situada en un servidor de datos DB2 utilizando datos textuales XML. Se pueden actualizar o insertar datos en columnas XML de una tabla usando datos XML binarios (datos que están en el formato Extensible Dynamic Binary XML DB2 Client/Server Binary XML) si el servidor de datos da soporte a los datos XML binarios.

Los tipos de datos de expresión de lenguaje principal que se pueden utilizar para actualizar las columnas XML son:
- java.sql.SQLXML (necesita un SDK para Java Versión 6 o posterior, y el controlador IBM Data Server Driver para JDBC y SQLJ versión 4.0 o posterior)
- com.ibm.db2.jcc.DB2Xml (en desuso)
- Serie
- byte
- Blob
- Clob
- sqlj.runtime.AsciiStream
- sqlj.runtime.BinaryStream
- sqlj.runtime.CharacterStream

La codificación de los datos XML se puede obtener a partir de los propios datos, lo cual se conoce como datos *codificados internamente*, o a partir de fuentes externas, lo cual se conoce como datos *codificados externamente*. Los datos XML que se envían al servidor de bases de datos como datos binarios se tratan como datos codificados internamente. Los datos XML que se envían a la fuente de datos como datos de tipo carácter se tratan como datos codificados externamente. Para JVM, se utiliza la codificación externa por omisión.

La codificación externa utilizada para aplicaciones Java es siempre la codificación Unicode.

Los datos codificados externamente pueden tener una codificación interna. Es decir, los datos se pueden enviar a la fuente de datos como datos de tipo carácter, pero los datos contienen información de codificación. La fuente de datos trata las incompatibilidades entre la codificación interna y la codificación externa de la manera siguiente:
- Si la fuente de datos es DB2 Database para Linux, UNIX y Windows, la fuente de datos genera un error si las codificaciones externa e interna son

incompatibles, a menos que ambas codificaciones sean Unicode. Si las codificaciones externa e interna son Unicode, la fuente de datos no tiene en cuenta la codificación interna.

- Si la fuente de datos es DB2 para z/OS, la fuente de datos no tiene en cuenta la codificación interna.

Los datos de las columnas XML se almacenan utilizando la codificación UTF-8.

Ejemplo: suponga que utiliza la sentencia siguiente para insertar datos de la expresión de lenguaje principal xmlString de tipo String en una columna XML de una tabla. xmlString es un tipo de caracteres, por lo que se utiliza su codificación externa, sin importar si tiene o no una especificación de codificación interna.

```
#sql [ctx] {INSERT INTO CUSTACC VALUES (1, :xmlString)};
```

Ejemplo: suponga que copia los datos de xmlString en una matriz de bytes con la codificación CP500. Los datos contienen una declaración XML con una declaración de codificación para CP500. Luego inserta los datos de la expresión de lenguaje principal byte[] en una columna XML de una tabla.

```
byte[] xmlBytes = xmlString.getBytes("CP500");
#sql[ctx] {INSERT INTO CUSTACC VALUES (4, :xmlBytes)};
```

Se considera que los datos de una serie de bytes se han codificado internamente. Los datos se convierten desde su sistema de codificación interna a UTF-8, si es necesario, y se almacenan en la fuente de datos según su formato jerárquico.

Ejemplo: suponga que copia los datos de xmlString en una matriz de bytes con la codificación US-ASCII. Luego crea una expresión de lenguaje principal sqlj.runtime.AsciiStream e inserta datos de esa expresión en una columna XML de una tabla de una fuente de datos.

```
byte[] b = xmlString.getBytes("US-ASCII");
java.io.ByteArrayInputStream xmlAsciiInputStream =
  new java.io.ByteArrayInputStream(b);
sqlj.runtime.AsciiStream sqljXmlAsciiStream =
  new sqlj.runtime.AsciiStream(xmlAsciiInputStream,  b.length);
#sql[ctx] {INSERT INTO CUSTACC VALUES (4, :sqljXmlAsciiStream)};
```

sqljXmlAsciiStream es un tipo de corriente, por lo cual, se utiliza su codificación interna. Los datos se convierten desde su codificación interna a UTF-8 y se almacenan en la fuente de datos según su formato jerárquico.

Ejemplo: expresión de lenguaje principal sqlj.runtime.CharacterStream: Suponga que crea la expresión de lenguaje principal sqlj.runtime.CharacterStream e inserta datos de esa expresión en una columna XML de una tabla.

```
java.io.StringReader xmlReader =
  new java.io.StringReader(xmlString);
sqlj.runtime.CharacterStream sqljXmlCharacterStream =
  new sqlj.runtime.CharacterStream(xmlReader,  xmlString.length());
#sql [ctx] {INSERT INTO CUSTACC VALUES (4, :sqljXmlCharacterStream)};
```

sqljXmlCharacterStream es un tipo de carácter, por lo cual, se utiliza su codificación externa, independientemente de que tenga una especificación de codificación interna.

Ejemplo: Suponga que recupera un documento de una columna XML y lo coloca en la expresión de lenguaje principal java.sql.SQLXML, e inserta los datos en una columna XML de una tabla.

```
java.sql.ResultSet rs = s.executeQuery ("SELECT * FROM CUSTACC");
rs.next();
java.sql.SQLXML xmlObject = (java.sql.SQLXML)rs.getObject(2);
#sql [ctx] {INSERT INTO CUSTACC VALUES (6, :xmlObject)};
```

Una vez que haya recuperado los datos, seguirán estando codificados en UTF-8. En consecuencia, cuando inserte los datos en otra columna XML, no se llevará a cabo ninguna conversión.

Ejemplo: Suponga que recupera un documento de una columna XML y lo coloca en la expresión de lenguaje principal com.ibm.db2.jcc.DB2Xml, e inserta los datos en una columna XML de una tabla.

```
java.sql.ResultSet rs = s.executeQuery ("SELECT * FROM CUSTACC");
rs.next();
com.ibm.db2.jcc.DB2Xml xmlObject = (com.ibm.db2.jcc.DB2Xml)rs.getObject(2);
#sql [ctx] {INSERT INTO CUSTACC VALUES (6, :xmlObject)};
```

Una vez que haya recuperado los datos, seguirán estando codificados en UTF-8. En consecuencia, cuando inserte los datos en otra columna XML, no se llevará a cabo ninguna conversión.

Recuperación de datos XML en aplicaciones de SQLJ

Cuando recupera datos procedentes de columnas XML de una tabla de base de datos en una aplicación SQLJ, los datos de salida se deben serializar explícita o implícitamente.

Los tipos de datos de iterador o expresión de lenguaje principal que puede utilizar para recuperar datos de las columnas XML son:

- java.sql.SQLXML (necesita un SDK para Java Versión 6 o posterior, y el controlador IBM Data Server Driver para JDBC y SQLJ versión 4.0 o posterior)
- com.ibm.db2.jcc.DB2Xml (en desuso)
- Serie
- byte[]
- sqlj.runtime.AsciiStream
- sqlj.runtime.BinaryStream
- sqlj.runtime.CharacterStream

Si la aplicación no llama a la función XMLSERIALIZE antes de la recuperación de datos, los datos se convierten de UTF-8 a la codificación de aplicación externa para los tipos de datos de caracteres o la codificación interna para los tipos de datos binarios. No se ha añadido una declaración XML. Si la expresión de lenguaje principal es un objeto de tipo java.sql.SQLXML o com.ibm.db2.jcc.DB2Xml, es necesario invocar un método adicional para recuperar los datos de este objeto. El método que invoque determina la codificación de los datos de salida y si se añade una declaración XML con una especificación de la codificación.

La tabla siguiente muestra los métodos que puede invocar para recuperar datos de un objeto java.sql.SQLXML o com.ibm.db2.jcc.DB2Xml, los tipos de datos de salida correspondientes y el tipo de codificación utilizado en las declaraciones XML.

Tabla 28. Métodos SQLXML y DB2Xml, tipos de datos y especificaciones de codificación añadidas

Método	Tipo de datos de salida	Tipo de declaración de codificación interna XML añadida
SQLXML.getBinaryStream	InputStream	Ninguno
SQLXML.getCharacterStream	Reader	Ninguno

Tabla 28. Métodos SQLXML y DB2Xml, tipos de datos y especificaciones de codificación añadidas (continuación)

Método	Tipo de datos de salida	Tipo de declaración de codificación interna XML añadida
SQLXML.getSource	Source	Ninguno
SQLXML.getString	Serie	Ninguno
DB2Xml.getDB2AsciiStream	InputStream	Ninguno
DB2Xml.getDB2BinaryStream	InputStream	Ninguno
DB2Xml.getDB2Bytes	byte[]	Ninguno
DB2Xml.getDB2CharacterStream	Reader	Ninguno
DB2Xml.getDB2String	Serie	Ninguno
DB2Xml.getDB2XmlAsciiStream	InputStream	US-ASCII
DB2Xml.getDB2XmlBinaryStream	InputStream	Especificado por el parámetro getDB2XmlBinaryStream *codificaciónDestino*
DB2Xml.getDB2XmlBytes	byte[]	Especificado por el parámetro DB2Xml.getDB2XmlBytes *codificaciónDestino*
DB2Xml.getDB2XmlCharacterStream	Reader	ISO-10646-UCS-2
DB2Xml.getDB2XmlString	Serie	ISO-10646-UCS-2

Si la aplicación ejecuta la función XMLSERIALIZE en los datos que se deben devolver, después de la ejecución de la función, los datos tendrán el tipo especificado en la función XMLSERIALIZE, no el tipo de datos XML. Por consiguiente, el controlador maneja los datos como el tipo especificado y pasa por alto cualquier declaración de codificación interna.

Ejemplo: Suponga que recupera datos de una columna XML en una expresión de lenguaje principal de tipo String.

```
#sql iterator  XmlStringIter (int, String);
#sql [ctx] siter = {SELECT C1, CADOC from CUSTACC};
#sql {FETCH :siter INTO :row, :outString};
```

String es un tipo de caracteres, de manera que los datos se convierten de UTF-8 a la codificación externa, la cual es la codificación JVM por omisión, y se devuelve sin ninguna declaración XML.

Ejemplo: Suponga que recupera datos de una columna XML en una expresión de lenguaje principal de tipo byte[].

```
#sql iterator  XmlByteArrayIter (int, byte[]);
XmlByteArrayIter biter = null;
#sql [ctx] biter =  {SELECT c1, CADOC from CUSTACC};
#sql {FETCH :biter INTO :row, :outBytes};
```

El tipo byte[] es un tipo binario, por lo que no se produce ninguna conversión de datos desde la codificación UTF-8, y los datos se devuelven sin ninguna declaración XML.

Ejemplo: Suponga que recupera un documento de una columna XML y lo coloca en la expresión de lenguaje principal java.sql.SQLXML, pero necesita los datos en una corriente binaria.

```
#sql iterator  SqlXmlIter (int, java.sql.SQLXML);
SqlXmlIter SQLXMLiter = null;
java.sql.SQLXML outSqlXml = null;
```

```
#sql [ctx] SqlXmlIter = {SELECT c1, CADOC from CUSTACC};
#sql {FETCH :SqlXmlIter INTO :row, :outSqlXml};
java.io.InputStream XmlStream = outSqlXml.getBinaryStream();
```

La sentencia FETCH recupera los datos y los coloca en el objeto SQLXML según la codificación UTF-8. SQLXML.getBinaryStream almacena los datos en una corriente binaria.

Ejemplo: Suponga que recupera un documento de una columna XML en la expresión de lenguaje principal com.ibm.db2.jcc.DB2Xml, pero necesita los datos de una serie de bytes con una declaración XML que incluya una especificación de codificación interna para UTF-8.

```
#sql iterator  DB2XmlIter (int, com.ibm.db2.jcc.DB2Xml);
DB2XmlIter db2xmliter = null;
com.ibm.db2.jcc.DB2Xml outDB2Xml = null;
#sql [ctx] db2xmliter = {SELECT c1, CADOC from CUSTACC};
#sql {FETCH :db2xmliter INTO :row, :outDB2Xml};
byte[] byteArray = outDB2XML.getDB2XmlBytes("UTF-8");
```

La sentencia FETCH recupera los datos en el objeto DB2Xml con codificación UTF-8. El método getDB2XmlBytes con el argumento UTF-8 añade una declaración XML con una especificación de codificación UTF-8 y almacena los datos en una matriz de bytes.

XMLCAST en aplicaciones SQLJ

Para utilizar XMLCAST a fin de convertir una variable de lenguaje principal al tipo de datos XML en una aplicación SQLJ, es necesario que convierta la variable de lenguaje principal al tipo de datos de SQL correspondiente.

Ejemplo: el código siguiente muestra una situación en la que es necesario convertir una variable de lenguaje principal de tipo String a datos SQL de tipo carácter, tal VARCHAR, para utilizar XMLCAST a fin de convertir el valor al tipo de datos XML.

```
String xmlresult = null;
String varchar_hv = "San Jose";
...
#sql [con] {SELECT XMLCAST(CAST(:varchar_hv AS VARCHAR(32)) AS XML) INTO
  :xmlresult FROM SYSIBM.SYSDUMMY1};
```

Utilización en SQLJ de funciones del SDK de Java Versión 5

Las aplicaciones SQLJ pueden utilizar varias funciones que aparecieron por primera vez en el SDK de Java Versión 5.

Importación estática

La importación estática le permite acceder a miembros estáticos sin calificarlos con el nombre de la clase a la que pertenecen. Para las aplicaciones SQLJ, esto significa que puede utilizar miembros estáticos en expresiones de lenguaje principal sin calificarlos.

Ejemplo: suponga que desea declarar una expresión de lenguaje principal que tiene este formato:

```
double r = cos(PI * E);
```

cos, PI y E son miembros de la clase java.lang.Math. Para declarar r sin calificar explícitamente cos, PI y E, incluya la siguiente sentencia de importación estática en su programa:

```
import static java.lang.Math.*;
```

Anotaciones

Las anotaciones Java son una forma de añadir metadatos a programas Java que también puede afectar a la forma en que esos programas son tratados por las herramientas y bibliotecas. Las anotaciones se declaran con declaraciones de tipo anotación, que son similares a las declaraciones de interfaz. Las anotaciones Java pueden aparecer en los tipos siguientes de clases o interfaces:

- Declaración de clase
- Declaración de interfaz
- Declaración de clase anidada
- Declaración de interfaz anidada

No puede incluir anotaciones Java directamente en los programas SQLJ, pero puede incluirlas en código fuente Java y luego incluir ese código fuente en los programas SQLJ.

Ejemplo: suponga que declara la siguiente anotación de marcador en un programa llamado MyAnnot.java:

```
public @interface MyAnot { }
```

También declara la siguiente anotación de marcador en un programa llamado MyAnnot2.java:

```
public @interface MyAnot2 { }
```

Puede luego utilizar esas anotaciones en un programa SQLJ:

```
// Anotaciones de clase
@MyAnot2 public @MyAnot class  TestAnnotation
{
  // Anotación de campo
  @MyAnot
  private static final int field1 = 0;
  // Anotación de constructor
  @MyAnot2 public @MyAnot TestAnnotation  () { }
   // Anotación de método
   @MyAnot
   public static void main (String a[])
   {
      TestAnnotation TestAnnotation_o = new TestAnnotation();
      TestAnnotation_o.runThis();
   }
  // Anotación de clase interna
  public static @MyAnot class TestAnotherInnerClass { }
  // Anotación de interfaz interna
  public static @MyAnot interface TestAnotInnerInterface { }
}
```

Tipos enumerados

Un tipo enumerado es un tipo de datos que consta de un conjunto de valores ordenados. El SDK de Java versión 5 proporciona como novedad el tipo `enum` para los tipos enumerados.

Puede incluir enums en los lugares siguientes:

- En archivos fuente Java (archivos .java) que incluya en un programa SQLJ
- En declaraciones de clases de SQLJ

Ejemplo: la declaración de clase TestEnum.sqlj incluye un tipo `enum`:

```
public class TestEnum2
{
  public enum Color {
   RED,ORANGE,YELLOW,GREEN,BLUE,INDIGO,VIOLET}
  Color color;
  ...                          // Obtener el valor del color
  switch (color) {
 case RED:
    System.out.println("Red is at one end of the spectrum.");
    #sql[ctx] { INSERT INTO MYTABLE VALUES (:color) };
   break;
 case VIOLET:
    System.out.println("Violet is on the other end of the spectrum.");
   break;
 case ORANGE:
 case YELLOW:
 case GREEN:
  case BLUE:
  case INDIGO:
    System.out.println("Everything else is in the middle.");
   break;
}
```

Valores genéricos

Puede utilizar valores genéricos en los programas Java para asignar un tipo a una colección Java. El programa traductor de SQLJ es compatible con la sintaxis de Java para valores genéricos. Esto son ejemplos de valores genéricos que puede utilizar en programas SQLJ:

- Una lista (List) de objetos List:

  ```
  List <List<String>> strList2 = new ArrayList<List<String>>();
  ```

- Un HashMap cuyo par clave/valor es de tipo String:

  ```
  Map <String,String> map = new HashMap<String,String>();
  ```

- Un método que utiliza como entrada una lista (List) con elementos de cualquier tipo:

  ```
  public void mthd(List <?> obj) {
  ...
  }
  ```

Aunque puede utilizar valores genéricos en variables de lenguaje principal de SQLJ, la utilidad de hacer esto es limitada porque el programa traductor de SQLJ no puede determinar los tipos de esas variables de lenguaje principal.

Bucle `for` mejorado

El bucle `for` mejorado le permite especificar la ejecución de un conjunto de operaciones para todos los miembros de una colección o matriz. Puede utilizar el iterador del bucle `for` mejorado en expresiones de lenguaje principal.

Ejemplo: la sentencia siguiente inserta cada elemento de la matriz names en la tabla TAB.

```
String[] names = {"ABC","DEF","GHI"};
for (String n : names)
{
  #sql {INSERT INTO TAB (VARCHARCOL) VALUES(:n) };
}
```

Varargs

Varargs facilita el pase de un número arbitrario de valores a un método. Un Vararg situado en la última posición de argumento de una declaración de método indica que los últimos argumentos son una matriz o secuencia de argumentos. Los argumentos pasados pueden ser utilizados por un programa SQLJ en expresiones de lenguaje principal.

Ejemplo: este ejemplo muestra el pase de un número arbitrario de parámetros de tipo Object a un método que inserta cada valor de parámetro en la tabla TAB.

```
public void runThis(Object...  objects) throws SQLException
{
  for (Object obj : objects)
  {
    #sql { INSERT INTO TAB (VARCHARCOL) VALUES(:obj) };
  }
}
```

Control de transacciones en aplicaciones SQLJ

En las aplicaciones SQLJ, al igual que en otros tipos de aplicaciones SQL, el control de transacciones supone la confirmación y retrotracción explícita o implícita de transacciones, y definir el nivel de aislamiento de las transacciones.

Establecimiento del nivel de aislamiento para una transacción SQLJ

Para establecer el nivel de aislamiento de una unidad de trabajo dentro de un programa SQLJ, utilice la cláusula SET TRANSACTION ISOLATION LEVEL.

Acerca de esta tarea

La tabla siguiente muestra los valores que puede especificar en la cláusula SET TRANSACTION ISOLATION LEVEL y sus valores equivalentes en DB2.

Tabla 29. Niveles de aislamiento equivalentes en SQLJ y DB2

Valor de SET TRANSACTION	Nivel de aislamiento en DB2
SERIALIZABLE	Lectura repetible
REPEATABLE READ	Estabilidad de lectura
READ COMMITTED	Estabilidad del cursor
READ UNCOMMITTED	Lectura no confirmada

El nivel de aislamiento afecta a la conexión JDBC subyacente así como a la conexión SQLJ.

Confirmación o retrotracción de transacciones SQLJ

Si inhabilita la confirmación automática para una conexión SQLJ, será necesario realizar operaciones explícitas de confirmación o retrotracción.

Acerca de esta tarea

Para ello utilizará cláusulas de ejecución que contienen sentencias COMMIT o ROLLBACK de SQL.

Ejemplo

Para confirmar una transacción de un programa SQLJ, utilice una sentencia como esta:

```
#sql [myConnCtx] {COMMIT};
```

Para retrotraer una transacción de un programa SQLJ, utilice una sentencia como esta:

```
#sql [myConnCtx] {ROLLBACK};
```

Manejo de errores y avisos de SQL en aplicaciones SQLJ

Las cláusulas de SQLJ emiten excepciones de SQL cuando se producen errores de SQL, pero no emiten excepciones para la mayoría de los avisos de SQL.

Acerca de esta tarea

SQLJ genera una excepción SQLException en las condiciones siguientes:

- Cuando una sentencia cualquiera de SQL devuelve un código de error de SQL negativo
- Cuando una sentencia SELECT INTO de SQL devuelve el código de error +100 de SQL

Para otros avisos de SQL, es necesario que compruebe explícitamente su existencia.

Procedimiento

- Para el manejo de errores de SQL, incluya bloques try/catch alrededor de las sentencias de SQLJ.
- Para el manejo de avisos de SQL, invoque el método getWarnings después de cada sentencia de SQLJ.

Manejo de errores de SQL en una aplicación SQLJ

Las cláusulas de SQLJ utilizan la clase java.sql.SQLException de JDBC para el manejo de errores.

Acerca de esta tarea

Para manejar errores de SQL en aplicaciones SQLJ, siga estos pasos:

Procedimiento

1. Importe la clase java.sql.SQLException.
2. Utilice los bloques `try/catch` de manejo de errores de Java para modificar el flujo del programa cuando se produzca un error de SQL.
3. Obtenga información sobre el error a partir de SQLException.

   Puede utilizar el método getErrorCode para obtener los códigos de error SQL, y el método getSQLState para obtener los SQLSTATE (estados de SQL).

Si está utilizando IBM Data Server Driver para JDBC y SQLJ, obtenga información adicional del SQLException convirtiéndolo en un objeto DB2Diagnosable, de la misma manera que obtiene esta información en una aplicación JDBC.

Ejemplo

El código de programa siguiente muestra el error de SQL que se produce si falla una sentencia SELECT.

```
try {
  #sql [ctxt] {SELECT LASTNAME INTO :empname
    FROM EMPLOYEE WHERE EMPNO='000010'};
}
catch (SQLException e) {
  System.out.println("Código de error devuelto: " + e.getErrorCode());
}
```

Manejo de avisos de SQL en una aplicación SQLJ

Salvo el código de error +100 de SQL para una sentencia SELECT INTO, los avisos del servidor de datos no emiten SQLExceptions. Para gestionar los avisos del servidor de datos, necesita otorgar al programa acceso a la clase java.sql.SQLWarning.

Acerca de esta tarea

Si desea recuperar información específica del servidor de datos sobre un aviso, también necesita otorgar al programa acceso a la interfaz com.ibm.db2.jcc.DB2Diagnosable y la clase com.ibm.db2.jcc.DB2Sqlca. Luego siga estos pasos:

Procedimiento

1. Configure un contexto de ejecución para esa cláusula de SQL. Consulte "Control de ejecución de las sentencias de SQL en SQLJ" para obtener información sobre cómo configurar un contexto de ejecución.
2. Para comprobar la existencia de un aviso del servidor de datos, invoque el método getWarnings después de ejecutar una cláusula SQLJ.

   getWarnings devuelve el primer objeto SQLWarning generado por una sentencia de SQL. Los objetos SQLWarning subsiguientes se encadenan al primero de ellos.
3. Para recuperar información específica del servidor de datos del objeto SQLWarning mediante IBM Data Server Driver para JDBC y SQLJ, siga las instrucciones de "Manejo de una excepción de SQL cuando se utiliza el IBM Data Server Driver para JDBC y SQLJ".

Ejemplo

El ejemplo siguiente muestra cómo recuperar un objeto SQLWarning para una cláusula de SQL cuyo contexto de ejecución es execCtx. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
ExecutionContext execCtx=myConnCtx.getExecutionContext();   1
                                    // Obtener contexto ejecución por omisión
                                    // del contexto de conexión
SQLWarning sqlWarn;
...
#sql [myConnCtx,execCtx] {SELECT LASTNAME INTO :empname
```

```
  FROM EMPLOYEE WHERE EMPNO='000010'};
if ((sqlWarn = execCtx.getWarnings()) != null)              2
System.out.println("SQLWarning " + sqlWarn);
```

Cierre de una conexión a una fuente de datos en una aplicación SQLJ

Una vez finalizada una conexión con una fuente de datos, es necesario que cierre la conexión con la fuente de datos. De esta forma se liberan inmediatamente los recursos de DB2 y SQLJ del objeto ConnectionContext asociado.

Acerca de esta tarea

Si no cierra un objeto ConnectionContext tras utilizarlo, pueden producirse comportamientos inesperados si un finalizador de Java cierra el objeto ConnectionContext. Algunos ejemplos de comportamiento inesperado son:

- Una ObjectClosedException en los objetos ResultSet o Statement subyacentes
- Un agente se cuelga en los procedimientos almacenados de DB2

Procedimiento

Para cerrar la conexión a la fuente de datos, utilice uno de los métodos ConnectionContext.close.

- Si ejecuta ConnectionContext.close() o ConnectionContext.close(ConnectionContext.CLOSE_CONNECTION), se cerrará el contexto de conexión y la conexión a la fuente de datos.
- Si ejecuta ConnectionContext.close(ConnectionContext.KEEP_CONNECTION) se cerrará el contexto de conexión, no así la conexión a la fuente de datos.

Ejemplo

El código de programa siguiente cierra el contexto de conexión, pero no cierra la conexión con la fuente de datos.

```
...
ctx = new EzSqljctx(con0);          // Crear un objeto de contexto de conexión
                                    // a partir de la conexión JDBC con0
...                                   // Ejecutar diversas operaciones de SQL
 EzSqljctx.close(ConnectionContext.KEEP_CONNECTION);
                                    // Cerrar el contexto de conexión pero mantener
                                    // abierta la conexión a la fuente de datos
```

Capítulo 5. Seguridad cuando se utiliza IBM Data Server Driver para JDBC y SQLJ

Cuando utiliza IBM Data Server Driver para JDBC y SQLJ, ha de seleccionar un mecanismo de seguridad especificando un valor para la propiedad securityMechanism de `Connection` o `DataSource`, o la propiedad de configuración global db2.jcc.securityMechanism.

Puede establecer la propiedad securityMechanism de una de las formas siguientes:

- Si utiliza la interfaz DriverManager, defina `securityMechanism` en un objeto java.util.Properties antes de invocar la modalidad del método getConnection que hace uso del parámetro java.util.Properties.
- Si utiliza la interfaz DataSource, y está creando y desplegando sus propios objetos DataSource, invoque el método DataSource.setSecurityMechanism después de crear un objeto DataSource.

Puede determinar el mecanismo de seguridad que está en vigor para una conexión invocando el método DB2Connection.getDB2SecurityMechanism.

La tabla siguiente lista los mecanismos de seguridad que se pueden utilizar con IBM Data Server Driver para JDBC y SQLJ, y las fuentes de datos que son compatibles con esos mecanismos de seguridad.

Tabla 30. Soporte de servidor de datos para los mecanismos de seguridad de IBM Data Server Driver para JDBC y SQLJ

Mecanismo de seguridad	Soportado por DB2 Database para Linux, UNIX y Windows	Soportado por DB2 para z/OS	Soportado por IBM Informix	Soportado por DB2 para i
ID de usuario y contraseña	Sí	Sí	Sí	Sí
ID de usuario solamente	Sí	Sí	Sí	Sí
ID de usuario y contraseña cifrada[1]	Sí	Sí	Sí	Sí[3]
ID de usuario cifrado[1]	Sí	Sí	No	No
ID de usuario cifrado y contraseña cifrada[1]	Sí	Sí	Sí	Sí[3]
ID de usuario cifrado y datos cifrados sensibles a la seguridad[1]	No	Sí	No	No
ID de usuario cifrado, contraseña cifrada y datos cifrados sensibles a la seguridad[1]	Sí	Sí	No	No
Kerberos[2]	Sí	Sí	No	Sí
Plugin[2]	Sí	No	No	No
Autentificación de certificado[2]	No	Sí	No	No

Tabla 30. Soporte de servidor de datos para los mecanismos de seguridad de IBM Data Server Driver para JDBC y SQLJ (continuación)

Mecanismo de seguridad	Soportado por DB2 Database para Linux, UNIX y Windows	Soportado por DB2 para z/OS	Soportado por IBM Informix	Soportado por DB2 para i

Nota:

1. Estos mecanismos de seguridad utilizan cifrado DRDA. El cifrado DRDA no tiene como finalidad proporcionar confidencialidad e integridad de contraseñas o datos a través de una red que no es segura, Internet. El cifrado DRDA utiliza un intercambio de claves anónimo, Diffie-Hellman, que no proporciona autentificación del servidor o del cliente. El cifrado DRDA es vulnerable ante ataques de intrusos.
2. Disponible solo para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4.
3. La versión de la fuente de datos debe ser DB2 para i V6R1 o posterior.

La tabla siguiente lista los mecanismos de seguridad que se pueden utilizar con IBM Data Server Driver para JDBC y SQLJ, y el valor que debe especificar para la propiedad `securityMechanism` correspondiente a cada mecanismo de seguridad.

El mecanismo de seguridad por omisión es CLEAR_TEXT_PASSWORD_SECURITY. Si el servidor no admite CLEAR_TEXT_PASSWORD_SECURITY pero sí admite ENCRYPTED_USER_AND_PASSWORD_SECURITY, IBM Data Server Driver para JDBC y SQLJ actualiza el mecanismo de seguridad a ENCRYPTED_USER_AND_PASSWORD_SECURITY y trata de conectarse al servidor. Cualquier otra discrepancia en el soporte del mecanismo de seguridad entre el solicitante y el servidor da como resultado un error.

Tabla 31. Mecanismos de seguridad soportados por IBM Data Server Driver para JDBC y SQLJ

Mecanismo de seguridad	Valor de la propiedad **securityMechanism**
ID de usuario y contraseña	`DB2BaseDataSource.CLEAR_TEXT_PASSWORD_SECURITY`
ID de usuario solamente	`DB2BaseDataSource.USER_ONLY_SECURITY`
ID de usuario y contraseña cifrada[1]	`DB2BaseDataSource.ENCRYPTED_PASSWORD_SECURITY`
ID de usuario cifrado[1]	`DB2BaseDataSource.ENCRYPTED_USER_ONLY_SECURITY`
ID de usuario cifrado y contraseña cifrada[1]	`DB2BaseDataSource.ENCRYPTED_USER_AND_PASSWORD_SECURITY`
ID de usuario cifrado y datos cifrados sensibles a la seguridad[1]	`DB2BaseDataSource.ENCRYPTED_USER_AND_DATA_SECURITY`
ID de usuario cifrado, contraseña cifrada y datos cifrados sensibles a la seguridad[1]	`DB2BaseDataSource.ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY`
Kerberos	`DB2BaseDataSource.KERBEROS_SECURITY`
Plugin	`DB2BaseDataSource.PLUGIN_SECURITY`
Autentificación de certificado	`DB2BaseDataSource.TLS_CLIENT_CERTIFICATE_SECURITY`

Nota:

1. El cifrado DRDA no tiene como finalidad proporcionar confidencialidad e integridad de contraseñas o datos a través de una red que no es segura, Internet. El cifrado DRDA utiliza un intercambio de claves anónimo, Diffie-Hellman, que no proporciona autentificación del servidor o del cliente. El cifrado DRDA es vulnerable ante ataques de intrusos.

La tabla siguiente muestra los tipos posibles de autentificación del servidor DB2 Database para Linux, UNIX y Windows y los valores compatibles de la propiedad securityMechanism de IBM Data Server Driver para JDBC y SQLJ.

Tabla 32. Tipos de autentificación del servidor DB2 Database para Linux, UNIX y Windows y valores de securityMechanism de IBM Data Server Driver para JDBC y SQLJ compatibles

Tipo de autentificación del servidor DB2 Database para Linux, UNIX y Windows	Valor de securityMechanism
CLIENT	USER_ONLY_SECURITY
SERVER	CLEAR_TEXT_PASSWORD_SECURITY
SERVER_ENCRYPT	CLEAR_TEXT_PASSWORD_SECURITY, ENCRYPTED_PASSWORD_SECURITY, o ENCRYPTED_USER_AND_PASSWORD_SECURITY
DATA_ENCRYPT	ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY
KERBEROS	KERBEROS_SECURITY o PLUGIN_SECURITY[2]
KRB_SERVER_ENCRYPT	KERBEROS_SECURITY , PLUGIN_SECURITY[1], ENCRYPTED_PASSWORD_SECURITY o ENCRYPTED_USER_AND_PASSWORD_SECURITY
GSSPLUGIN	PLUGIN_SECURITY[1] o KERBEROS_SECURITY
GSS_SERVER_ENCRYPT[3]	CLEAR_TEXT_PASSWORD_SECURITY, ENCRYPTED_PASSWORD_SECURITY, ENCRYPTED_USER_AND_PASSWORD_SECURITY, PLUGIN_SECURITY o KERBEROS_SECURITY

Notas:

1. Para PLUGIN_SECURITY, debe tratarse de un Kerberos.
2. Para PLUGIN_SECURITY, uno de los plugins del servidor se identifica a sí mismo con capacidad para dar soporte a Kerberos.
3. GSS_SERVER_ENCRYPT es una combinación de GSSPLUGIN y SERVER_ENCRYPT.

Seguridad basada en ID de usuario y contraseña cuando se utiliza IBM Data Server Driver para JDBC y SQLJ

Cuando se utiliza IBM Data Server Driver para JDBC y SQLJ, uno de los métodos de seguridad disponibles es la seguridad basada en ID de usuario y contraseña.

Para especificar la seguridad basada en un ID de usuario y una contraseña para una conexión JDBC, utilice una de las técnicas siguientes.

Para la interfaz DriverManager: puede especificar el ID de usuario y la contraseña directamente en la invocación de DriverManager.getConnection. Por ejemplo:

```
import java.sql.*;          // Base JDBC
...
String id = "dbadm";                    // Definir ID de usuario
String pw = "dbadm";                    // Definir contraseña
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                            // Establecer URL para la fuente de datos

Connection con = DriverManager.getConnection(url, id, pw);
                            // Crear conexión
```

Otro método consiste en definir el ID de usuario y la contraseña directamente en la serie de caracteres del URL. Por ejemplo:

```
import java.sql.*;         // Base JDBC
...
String url =
  "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose:user=dbadm;password=dbadm;";

                           // Establecer URL para la fuente de datos
Connection con = DriverManager.getConnection(url);
                           // Crear conexión
```

Como alternativa, puede establecer el ID de usuario y la contraseña definiendo las propiedades `user` y `password` en un objeto Properties, y luego invocar la modalidad del método getConnection que hace uso del objeto Properties como parámetro. Opcionalmente, puede definir la propiedad `securityMechanism` para indicar que está utilizando la seguridad basada en un ID de usuario y contraseña. Por ejemplo:

```
import java.sql.*;                          // Base JDBC
import com.ibm.db2.jcc.*;                // Implementación de IBM Data Server
                                            // Driver para JDBC y SQLJ de JDBC
...
Properties properties = new java.util.Properties();
                                            // Crear objeto Properties
properties.put("user", "dbadm");         // Definir ID de usuario para conexión
properties.put("password", "dbadm");     // Definir contraseña para conexión
properties.put("securityMechanism",
  new String("" + com.ibm.db2.jcc.DB2BaseDataSource.CLEAR_TEXT_PASSWORD_SECURITY +
  ""));
                                            // Establecer mecanismo de seguridad
                                            // en ID de usuario y contraseña
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                                            // Establecer URL para la fuente de datos
Connection con = DriverManager.getConnection(url, properties);
                                            // Crear conexión
```

Para la interfaz DataSource: puede especificar el ID de usuario y la contraseña directamente en la invocación de DataSource.getConnection. Por ejemplo:

```
import java.sql.*;                          // Base JDBC
import com.ibm.db2.jcc.*;                // Implementación de IBM Data Server
                                            // Driver para JDBC y SQLJ de JDBC
...
Context ctx=new InitialContext();        // Crear contexto para JNDI
DataSource ds=(DataSource)ctx.lookup("jdbc/sampledb");
                                            // Obtener objeto DataSource
String id = "dbadm";                     // Definir ID de usuario
String pw = "dbadm";                     // Definir contraseña
Connection con = ds.getConnection(id, pw);
                                            // Crear conexión
```

Como alternativa, si crea y despliega el objeto DataSource, puede establecer el ID de usuario y la contraseña invocando los métodos DataSource.setUser y DataSource.setPassword después de crear el objeto DataSource. Opcionalmente, puede invocar el método DataSource.setSecurityMechanism para indicar que está utilizando la seguridad basada en un ID de usuario y contraseña. Por ejemplo:

```
...
com.ibm.db2.jcc.DB2SimpleDataSource ds =  // Crear objeto DB2SimpleDataSource
  new com.ibm.db2.jcc.DB2SimpleDataSource();
ds.setDriverType(4);                       // Establecer tipo de controlador
ds.setDatabaseName("san_jose");            // Establecer ubicación
ds.setServerName("mvs1.sj.ibm.com");       // Establecer nombre de servidor
ds.setPortNumber(5021);                    // Establecer número de puerto
ds.setUser("dbadm");                       // Establecer ID de usuario
ds.setPassword("dbadm");                   // Establecer contraseña
ds.setSecurityMechanism(
```

```
  com.ibm.db2.jcc.DB2BaseDataSource.CLEAR_TEXT_PASSWORD_SECURITY);
                                                  // Establecer mecanismo de seguridad
                                                  // en ID de usuario y contraseña
```

Caracteres válidos en las contraseñas: Todos los caracteres comprendidos dentro del rango ASCII X'20' (decimal 32) a X'7E'' (decimal 126) son válidos en las contraseñas, excepto para los caracteres siguientes:

- X'20' (espacio) al final de una contraseña. IBM Data Server Driver para JDBC y SQLJ elimina los caracteres de espacio al final de una contraseña.
- X'3B' (punto y coma)
- Los caracteres que no pueden convertirse en caracteres EBCDIC, si las contraseñas en texto sin formato se envían a un servidor de datos.

Seguridad mediante frase de contraseña de RACF: si va a establecer conexión con un DB2 para z/OS configurado para la protección de RACF y la versión de RACF da soporte a las frases de contraseña de RACF, puede proporcionar una frase de contraseña de RACF para el valor de la propiedad password, en lugar de sólo una contraseña. Una frase de contraseña debe respetar las reglas siguientes:

- Una frase de contraseña es una serie de caracteres que consta de letras con combinación de mayúsculas y minúsculas, números y caracteres especiales, incluidos espacios en blanco.
- La longitud de la frase de contraseña puede ser de 9 a 100 caracteres o de 14 a 100 caracteres.

  Las frases de contraseña que tienen entre 9 y 13 caracteres se permiten cuando se instala la salida new-password-phrase (ICHPWX11) en el sistema z/OS y la salida permite las frases de contraseña de menos de 14 caracteres.
- Una frase de contraseña no debe contener el ID de usuario en formato de caracteres secuenciales ya sea en mayúsculas como en minúsculas.
- Una frase de contraseña debe contener, como mínimo, dos caracteres alfabéticos (de la A a la Z o de la a a la z).
- Una frase de contraseña debe contener, como mínimo, dos caracteres no alfabéticos (números, caracteres de puntuación o caracteres especiales).
- Una frase de contraseña no debe contener más de dos caracteres consecutivos que sean idénticos.
- Si hay una comilla simple (') en la frase de contraseña, la comilla simple debe representarse como dos comillas simples consecutivas ('').

En el ejemplo siguiente se utiliza una frase de contraseña para una conexión:

```
import java.sql.*;                          // Base JDBC
import com.ibm.db2.jcc.*;                // Implementación de IBM Data Server
                                            // Driver para JDBC y SQLJ de JDBC
...
Properties properties = new java.util.Properties();
                                            // Crear objeto Properties
properties.put("user", "dbadm");       // Definir ID de usuario para conexión
properties.put("password", "a*b!c@ D12345 678");
                                            // Establecer la frase de contraseña
                                            // para la conexión
properties.put("securityMechanism",
  new String("" + com.ibm.db2.jcc.DB2BaseDataSource.CLEAR_TEXT_PASSWORD_SECURITY +
  ""));
                                            // Establecer mecanismo de seguridad
                                            // en ID de usuario y contraseña
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
```

```
                                        // Establecer URL para la fuente de datos
Connection con = DriverManager.getConnection(url, properties);
                                        // Crear conexión
```

Seguridad mediante los ID de usuario cuando se utiliza IBM Data Server Driver para JDBC y SQLJ

Cuando se utiliza IBM Data Server Driver para JDBC y SQLJ, uno de los métodos de seguridad disponibles es la seguridad mediante los ID de usuario.

Para especificar la seguridad basada en un ID de usuario para una conexión JDBC, utilice una de las técnicas siguientes.

Para la interfaz DriverManager: establezca el ID de usuario y el mecanismo de seguridad definiendo las propiedades `user` y `securityMechanism` en un objeto Properties, y luego invoque la modalidad del método getConnection que hace uso del objeto Properties como parámetro. Por ejemplo:

```
import java.sql.*;                      // JDBC base
import com.ibm.db2.jcc.*;               // Implementación de IBM Data Server
                                        // Driver para JDBC y SQLJ
                                        // de JDBC
...
Properties properties = new Properties(); // Crear un objeto Properties
properties.put("user", "db2adm");         // Definir ID de usuario para la conexión
properties.put("securityMechanism",
  new String("" + com.ibm.db2.jcc.DB2BaseDataSource.USER_ONLY_SECURITY + ""));
                                        // Establecer mecanismo de seguridad
                                        // en ID de usuario solamente
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                                        // Establecer URL para la fuente de datos
Connection con = DriverManager.getConnection(url, properties);
                                        // Crear la conexión
```

Para la interfaz DataSource: si crea y despliega el objeto DataSource, establezca el ID de usuario y el mecanismo de seguridad invocando los métodos DataSource.setUser y DataSource.setSecurityMechanism después de crear el objeto DataSource. Por ejemplo:

```
import java.sql.*;                      // JDBC base
import com.ibm.db2.jcc.*;               // Implementación de IBM Data Server
                                        // Driver para JDBC y SQLJ
                                        // de JDBC
...
com.ibm.db2.jcc.DB2SimpleDataSource db2ds =
  new com.ibm.db2.jcc.DB2SimpleDataSource();
                                        // Crear objeto DB2SimpleDataSource
db2ds.setDriverType(4);                 // Definir tipo de controlador
db2ds.setDatabaseName("san_jose");      // Definir la ubicación
db2ds.setServerName("mvs1.sj.ibm.com");
                                        // Establecer nombre de servidor
db2ds.setPortNumber(5021);              // Definir número de puerto
db2ds.setUser("db2adm");                // Definir ID de usuario
db2ds.setSecurityMechanism(
  com.ibm.db2.jcc.DB2BaseDataSource.USER_ONLY_SECURITY);
                                        // Establecer mecanismo de seguridad
                                        // sólo de ID de usuario
```

Contraseña, ID de usuario o seguridad de datos cifrados en IBM Data Server Driver para JDBC y SQLJ

IBM Data Server Driver para JDBC y SQLJ da soporte al cifrado de los ID de usuario, contraseñas o datos cuando las aplicaciones Java acceden a los servidores de datos.

Esos mecanismos de seguridad utilizan cifrado DRDA. El cifrado DRDA no tiene como finalidad proporcionar confidencialidad e integridad de contraseñas o datos a través de una red que no es segura, Internet. El cifrado DRDA utiliza un intercambio de claves anónimo, Diffie-Hellman, que no proporciona autentificación del servidor o del cliente. El cifrado DRDA es vulnerable ante ataques de intrusos.

IBM Data Server Driver para JDBC y SQLJ es compatible con el cifrado DES de 56 bits (débil) y el cifrado AES de 256 bits (fuerte). El cifrado AES sólo está disponible para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4. Defina la propiedad encryptionAlgorithm del controlador para elegir entre el cifrado DES de 56 bits (valor 1 para encryptionAlgorithm) y el cifrado AES de 256 bits (valor 2 para encryptionAlgorithm). El cifrado AES de 256 bits se utiliza para una conexión solamente si el servidor de bases de datos es compatible con él y está configurado para utilizarlo.

Si utiliza la seguridad por contraseña cifrada, la seguridad por ID de usuario cifrado o la seguridad por ID de usuario cifrado y contraseña cifrada, es necesario que IBM Java Cryptography Extension (JCE) esté habilitado en el cliente. IBM JCE forma parte del SDK para Java de IBM, versión 1.4.2 o posterior.

IBM JCE necesita utilizar comunicaciones cliente/servidor cifradas por el algoritmo DES de 56 bits o AES de 256 bits entre IBM Data Server Driver para JDBC y SQLJ y servidores DB2 Database para Linux, UNIX y Windows.

Para el cifrado AES, necesita un archivo de política no restringida para JCE. Para IBM SDK para Java, el archivo está disponible en la ubicación siguiente:

```
https://www14.software.ibm.com/webapp/iwm/web/preLogin.do?source=jcesdk
```

Cuando utiliza cifrado AES con el SDK para Java desde Oracle, es necesario instalar el archivo JCE Unlimited Strength Jurisdiction Policy File. Este archivo está disponible en Oracle. Si no se encuentra el archivo JCE Unlimited Strength Jurisdiction Policy File, se emite una java.security.InvalidKeyException.

Las conexiones con servidores DB2 para i versión 6R1 o posterior pueden utilizar seguridad por contraseña cifrada o seguridad por ID de usuario cifrado y contraseña cifrada. Para la seguridad por contraseña cifrada o la seguridad por ID de usuario cifrado y contraseña cifrada, es necesario que IBM Java Cryptography Extension (ibmjceprovidere.jar) esté instalado en el cliente. IBM JCE forma parte del SDK para Java de IBM, versión 1.4.2 o posterior.

También puede utilizar los datos cifrados sensibles a la seguridad además de la seguridad de ID de usuario cifrado o la seguridad de ID de usuario cifrado y de contraseña cifrada. Debe especificar el cifrado de los datos sensibles a la seguridad mediante el valor `ENCRYPTED_USER_AND_DATA_SECURITY` o `ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY` de `securityMechanism`. `ENCRYPTED_USER_AND_DATA_SECURITY` es válido únicamente para conexiones con servidores DB2 para z/OS y únicamente para cifrado DES (valor 1 de encryptionAlgorithm).

Los servidores de bases de datos DB2 para z/OS o DB2 Database para Linux, UNIX y Windows cifran los datos siguientes cuando el usuario especifica que se realice el cifrado de datos que deben protegerse:
- Sentencias de SQL que se preparan, ejecutan o vinculan a un paquete
- Información de parámetros de entrada y de salida
- Conjuntos de resultados
- Datos LOB
- datos XML
- Resultados de operaciones de descripción

Antes de utilizar datos cifrados sensibles a la seguridad, z/OS Integrated Cryptographic Services Facility debe estar instalado y habilitado en el sistema operativo z/OS.

Para especificar el mecanismo de seguridad basado en un ID de usuario cifrado o contraseña cifrada para una conexión JDBC, utilice una de las técnicas siguientes.

Para la interfaz DriverManager: establezca el ID de usuario, la contraseña y el mecanismo de seguridad definiendo las propiedades `user`, `password` y `securityMechanism` en un objeto Properties, y luego invoque la modalidad del método getConnection que hace uso del objeto Properties como parámetro. Por ejemplo, utilice un código como el siguiente para establecer el ID de usuario cifrado, la contraseña cifrada y un mecanismo de datos sensibles a la seguridad cifrados, con cifrado AES:

```
import java.sql.*;                          // Base JDBC
import com.ibm.db2.jcc.*;                // Implementación de IBM Data Server
                                            // Driver para JDBC y SQLJ de JDBC
...
Properties properties = new Properties(); // Crear un objeto Properties
properties.put("user", "dbadm");       // Definir ID de usuario para conexión
properties.put("password", "dbadm");  // Definir contraseña para conexión
properties.put("securityMechanism",
  new String("" +
  com.ibm.db2.jcc.DB2BaseDataSource.ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY +
  ""));
                                            // Establecer mecanismo de seguridad
                                            // ID usuario y contraseña cifrada
properties.put("encryptionAlgorithm", "2");
                                            // Solicitar seguridad AES
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                                            // Establecer URL para la fuente de datos
Connection con = DriverManager.getConnection(url, properties);
                                            // Crear la conexión
```

Para la interfaz DataSource: si crea y despliega el objeto DataSource, puede definir el ID de usuario, contraseña y mecanismo de seguridad invocando los métodos DataSource.setUser, DataSource.setPassword y DataSource.setSecurityMechanism después de crear el objeto DataSource. Por ejemplo, utilice un código como el siguiente para establecer el ID de usuario cifrado, la contraseña cifrada y un mecanismo de seguridad de datos sensibles a la seguridad cifrados, con cifrado AES:

```
import java.sql.*;                          // Base JDBC
import com.ibm.db2.jcc.*;                // Implementación de IBM Data Server
                                            // Driver para JDBC y SQLJ de JDBC
...
com.ibm.db2.jcc.DB2SimpleDataSource ds =
  new com.ibm.db2.jcc.DB2SimpleDataSource();
                                            // Crear el objeto DataSource
ds.setDriverType(4);                 // Establecer el tipo de controlador
ds.setDatabaseName("san_jose");      // Establecer ubicación
```

```
ds.setServerName("mvs1.sj.ibm.com");
                                          // Establecer nombre de servidor
ds.setPortNumber(5021);              // Establecer número de puerto
ds.setUser("db2adm");                // Establecer el ID de usuario
ds.setPassword("db2adm");            // Establecer contraseña
ds.setSecurityMechanism(
  com.ibm.db2.jcc.DB2BaseDataSource.ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY);
                                          // Establecer mecanismo de seguridad
                                          // ID usuario y contraseña cifrada
ds.setEncryptionAlgorithm(2);             // Solicitar cifrado AES
```

Caracteres válidos en las contraseñas: Todos los caracteres comprendidos dentro del rango ASCII X'20' (decimal 32) a X'7E'' (decimal 126) son válidos en las contraseñas, excepto para los caracteres siguientes:

- X'20' (espacio) al final de una contraseña. IBM Data Server Driver para JDBC y SQLJ elimina los caracteres de espacio al final de una contraseña.
- X'3B' (punto y coma)
- Los caracteres que no pueden convertirse en caracteres EBCDIC, si las contraseñas en texto sin formato se envían a un servidor de datos.

Seguridad mediante frase de contraseña de RACF: si va a establecer conexión con un DB2 para z/OS configurado para la protección de RACF y la versión de RACF da soporte a las frases de contraseña de RACF, puede proporcionar una frase de contraseña de RACF para el valor de la propiedad password, en lugar de sólo una contraseña. Una frase de contraseña debe respetar las reglas siguientes:

- Una frase de contraseña es una serie de caracteres que consta de letras con combinación de mayúsculas y minúsculas, números y caracteres especiales, incluidos espacios en blanco.
- La longitud de la frase de contraseña puede ser de 9 a 100 caracteres o de 14 a 100 caracteres.

  Las frases de contraseña que tienen entre 9 y 13 caracteres se permiten cuando se instala la salida new-password-phrase (ICHPWX11) en el sistema z/OS y la salida permite las frases de contraseña de menos de 14 caracteres.
- Una frase de contraseña no debe contener el ID de usuario en formato de caracteres secuenciales ya sea en mayúsculas como en minúsculas.
- Una frase de contraseña debe contener, como mínimo, dos caracteres alfabéticos (de la A a la Z o de la a a la z).
- Una frase de contraseña debe contener, como mínimo, dos caracteres no alfabéticos (números, caracteres de puntuación o caracteres especiales).
- Una frase de contraseña no debe contener más de dos caracteres consecutivos que sean idénticos.
- Si hay una comilla simple (') en la frase de contraseña, la comilla simple debe representarse como dos comillas simples consecutivas ('').

Seguridad Kerberos cuando se utiliza IBM Data Server Driver para JDBC y SQLJ

El soporte de JDBC para la seguridad Kerberos sólo está disponible para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4.

Para habilitar el soporte de JDBC para la seguridad Kerberos, también necesita habilitar los componentes siguientes del kit de desarrollo de software (SDK) para Java:

- Java Cryptography Extension

- JGSS (servicio de seguridad genérica de Java)
- JAAS (servicio de autentificación y autorización de Java)

Consulte la documentación de SDK para Java si desea información sobre cómo habilitar estos componentes.

Existen tres formas de especificar la seguridad Kerberos para una conexión:
- Con un ID de usuario y contraseña
- Sin un ID de usuario ni contraseña
- Con una credencial delegada

Seguridad Kerberos con un ID de usuario y contraseña

Para este caso, Kerberos utiliza el ID de usuario y la contraseña especificados para obtener un TGT (certificado de otorgamiento de certificados) que permite autentificarse ante el servidor de bases de datos.

Es necesario que defina las propiedades `user`, `password`, `kerberosServerPrincipal` y `securityMechanism`. Defina la propiedad `securityMechanism` como com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY (11). La propiedad `kerberosServerPrincipal` especifica el nombre de principal que el servidor de bases de datos registra en un Centro de distribución de claves (KDC) de Kerberos.

Para la interfaz DriverManager: establezca el ID de usuario, la contraseña, el servidor Kerberos y el mecanismo de seguridad definiendo las propiedades `user`, `password`, `kerberosServerPrincipal` y `securityMechanism` en un objeto Properties, y luego invoque la modalidad del método getConnection que hace uso del objeto Properties como parámetro. Por ejemplo, utilice un código como el siguiente para establecer el mecanismo de seguridad Kerberos con un ID de usuario y contraseña:

```
import java.sql.*;                          // Base JDBC
import com.ibm.db2.jcc.*;                // Implementación de IBM Data Server
                                            // Driver para JDBC y SQLJ de JDBC
...
Properties properties = new Properties(); // Crear un objeto Properties
properties.put("user", "db2adm");         // Establecer ID usuario para la conexión
properties.put("password", "db2adm");     // Establecer contraseña para la conexión
properties.put("kerberosServerPrincipal",
  "sample/srvlsj.ibm.com@SRVLSJ.SJ.IBM.COM");
                                          // Establecer servidor Kerberos
properties.put("securityMechanism",
  new String("" +
  com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY + ""));
                                          // Establecer mecanismo de seguridad
                                          // Kerberos
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                                          // Establecer URL para la fuente de datos
Connection con = DriverManager.getConnection(url, properties);
                                          // Crear la conexión
```

Para la interfaz DataSource: si crea y despliega el objeto DataSource, establezca el servidor Kerberos y el mecanismo de seguridad invocando los métodos DataSource.setKerberosServerPrincipal y `DataSource.setSecurityMechanism` después de crear el objeto DataSource. Por ejemplo:

```
import java.sql.*;                          // Base JDBC
import com.ibm.db2.jcc.*;                // Implementación de IBM Data Server
                                            // Driver para JDBC y SQLJ de JDBC
...
com.ibm.db2.jcc.DB2SimpleDataSource db2ds =
  new com.ibm.db2.jcc.DB2SimpleDataSource();
                                          // Crear el objeto DataSource
```

```
db2ds.setDriverType(4);                    // Definir tipo de controlador
db2ds.setDatabaseName("san_jose");         // Definir la ubicación
db2ds.setUser("db2adm");                   // Definir ID de usuario
db2ds.setPassword("db2adm");               // Definir contraseña
db2ds.setServerName("mvs1.sj.ibm.com");
                                           // Establecer nombre de servidor
db2ds.setPortNumber(5021);                 // Definir número de puerto
db2ds.setKerberosServerPrincipal(
  "sample/srvlsj.ibm.com@SRVLSJ.SJ.IBM.COM");
                                           // Establecer servidor Kerberos
db2ds.setSecurityMechanism(
  com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY);
                                           // Establecer mecanismo de seguridad
                                           // Kerberos
```

Seguridad Kerberos sin ningún ID de usuario ni contraseña

Para este caso, la antememoria de credenciales por omisión de Kerberos debe contener un TGT (certificado de otorgamiento de certificados) que permita autentificarse ante el servidor de bases de datos.

Es necesario establecer las propiedades `kerberosServerPrincipal` y `securityMechanism`. Defina la propiedad `securityMechanism` como com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY (11).

Para la interfaz DriverManager: Defina el servidor Kerberos y el mecanismo de seguridad estableciendo las propiedades `kerberosServerPrincipal` y `securityMechanism` en un objeto Properties, e invocando a continuación el formato del método getConnection que incluye el objeto Properties como parámetro. Por ejemplo, utilice un código como el siguiente para establecer el mecanismo de seguridad Kerberos sin un ID de usuario ni contraseña:

```
import java.sql.*;                         // Base JDBC
import com.ibm.db2.jcc.*;                 // Implementación de IBM Data Server
                                           // Driver para JDBC y SQLJ de JDBC
...
Properties properties = new Properties(); // Crear un objeto Properties
properties.put("kerberosServerPrincipal",
  "sample/srvlsj.ibm.com@SRVLSJ.SJ.IBM.COM");
                                           // Establecer servidor Kerberos
properties.put("securityMechanism",
  new String("" +
  com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY + ""));
                                           // Establecer mecanismo de seguridad
                                           // Kerberos
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                                           // Establecer URL para la fuente de datos
Connection con = DriverManager.getConnection(url, properties);
                                           // Crear la conexión
```

Para la interfaz DataSource: si crea y despliega el objeto DataSource, establezca el servidor Kerberos y el mecanismo de seguridad invocando los métodos DataSource.setKerberosServerPrincipal y `DataSource.setSecurityMechanism` después de crear el objeto DataSource. Por ejemplo:

```
import java.sql.*;                         // Base JDBC
import com.ibm.db2.jcc.*;                 // Implementación de IBM Data Server
                                           // Driver para JDBC y SQLJ de JDBC
...
DB2SimpleDataSource db2ds =
  new com.ibm.db2.jcc.DB2SimpleDataSource();
                                           // Crear el objeto DataSource
db2ds.setDriverType(4);                    // Definir tipo de controlador
db2ds.setDatabaseName("san_jose");         // Definir la ubicación
```

```
db2ds.setServerName("mvs1.sj.ibm.com");
                                        // Establecer nombre de servidor
db2ds.setPortNumber(5021);              // Definir número de puerto
db2ds.setKerberosServerPrincipal(
  "sample/srvlsj.ibm.com@SRVLSJ.SJ.IBM.COM");
                                        // Establecer servidor Kerberos
db2ds.setSecurityMechanism(
  com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY);
                                        // Establecer mecanismo de seguridad
                                        // Kerberos
```

Seguridad Kerberos con una credencial delegada de otro principal

Para este caso, se autentifica ante el servidor de bases de datos mediante una credencial delegada que le pasa otro principal.

Es necesario establecer las propiedades `kerberosServerPrincipal`, `gssCredential` , y `securityMechanism`. Defina la propiedad `securityMechanism` como com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY (11).

Para la interfaz DriverManager: Defina el servidor Kerberos, la credencial delegada y el mecanismo de seguridad estableciendo las propiedades `kerberosServerPrincipal` y `securityMechanism` en un objeto Properties. A continuación, invoque el formato del método getConnection que incluye el objeto Properties como parámetro. Por ejemplo, utilice un código como el siguiente para establecer el mecanismo de seguridad Kerberos sin un ID de usuario ni contraseña:

```
import java.sql.*;                      // Base JDBC
import com.ibm.db2.jcc.*;              // Implementación de IBM Data Server
                                        // Driver para JDBC y SQLJ de JDBC
...
Properties properties = new Properties(); // Crear un objeto Properties
properties.put("kerberosServerPrincipal",
  "sample/srvlsj.ibm.com@SRVLSJ.SJ.IBM.COM");
                                        // Establecer servidor Kerberos
properties.put("gssCredential",delegatedCredential);
                                        // Establecer credencial delegada
properties.put("securityMechanism",
  new String("" +
    com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY + ""));
                                        // Establecer mecanismo de seguridad
                                        // Kerberos
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                                        // Establecer URL para la fuente de datos
Connection con = DriverManager.getConnection(url, properties);
                                        // Crear la conexión
```

Para la interfaz DataSource: Si crea y despliega el objeto DataSource, defina el servidor Kerberos, la credencial delegada y el mecanismo de seguridad invocando los parámetros DataSource.setKerberosServerPrincipal, DataSource.setGssCredential y DataSource.setSecurityMechanism después de crear el objeto DataSource. Por ejemplo:

```
DB2SimpleDataSource db2ds = new com.ibm.db2.jcc.DB2SimpleDataSource();
                                          // Crear el objeto DataSource
db2ds.setDriverType(4);                   // Definir tipo de controlador
db2ds.setDatabaseName("san_jose");        // Definir la ubicación
db2ds.setServerName("mvs1.sj.ibm.com");   // Definir el nombre de servidor
db2ds.setPortNumber(5021);                // Definir número de puerto
db2ds.setKerberosServerPrincipal(
  "sample/srvlsj.ibm.com@SRVLSJ.SJ.IBM.COM");
                                          // Establecer servidor Kerberos
db2ds.setGssCredential(delegatedCredential);
```

```
                                                // Establecer credencial delegada
db2ds.setSecurityMechanism(
  com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY);
                                                // Establecer mecanismo de seguridad
                                                // Kerberos
```

Soporte del plugin de seguridad de IBM Data Server Driver para JDBC y SQLJ

Puede crear mecanismos de autentificación propios en forma de bibliotecas de carga o plugins que DB2 Database para Linux, UNIX y Windows carga para llevar a cabo la autentificación del usuario. Para soportar el desarrollo de plugins de seguridad en Java, IBM Data Server Driver para JDBC y SQLJ proporciona soporte de plugin de seguridad.

El soporte para el plugin de seguridad de IBM Data Server Driver para JDBC y SQLJ está disponible solamente para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 con servidores DB2 Database para Linux, UNIX y Windows.

Para utilizar la seguridad por plugin, necesita un plugin de seguridad en el cliente y otro plugin en el servidor.

Los plugins de seguridad deben incluir los elementos siguientes:

- Una clase que amplía la clase abstracta com.ibm.db2.jcc.DB2JCCPlugin.

  La clase abstracta com.ibm.db2.jcc.DB2JCCPlugin se proporciona con IBM Data Server Driver para JDBC y SQLJ.
- En la clase com.ibm.db2.jcc.DB2JCCPlugin, un método com.ibm.db2.jcc.DB2JCCPlugin.getTicket

  Este método recupera un certificado Kerberos de un usuario y devuelve información de contexto de seguridad en una matriz de bytes. La información de la matriz de bytes es utilizada por IBM Data Server Driver para JDBC y SQLJ para acceder al servidor de bases de datos DB2.
- Implementaciones de varios métodos que se definen en las interfaces org.ietf.jgss.GSSContext y `org.ietf.jgss.GSSCredential`

  Estas implementaciones de métodos deben seguir las especificaciones siguientes: Generic Security Service Application Program Interface, Versión 2 (IETF RFC2743) y Generic Security Service API Versión 2: Java-Bindings (IETF RFC2853). El plugin debe implementar y llamar a los métodos siguientes:

  GSSContext.dispose
  : Libera cualquier información criptográfica y de recursos del sistema que se almacena en un objeto de contexto e invalida el contexto.

  GSSContext.getCredDelegState
  : Determina si se habilita la delegación de credenciales en un contexto.

  GSSContext.getMutualAuthState
  : Determina si se habilita la autentificación mutua en el contexto.

  GSSContext.initSecContext
  : Inicia la fase de creación del contexto y procesa todos los símbolos que se generan mediante el método acceptSecContext de la entidad homóloga.

GSSContext.requestCredDeleg
Solicita que las credenciales del iniciador se deleguen a la persona que acepte la credencial cuando se establezca una conexión.

GSSContext.requestMutualAuth
Solicita la autentificación mutua cuando se establece un contexto.

GSSCredential.dispose
Libera toda la información confidencial que contiene el objeto GSSCredential.

En sqllib/samples/java/jdbc, se proporcionan dos ejemplos de plugin de Java para ayudar al usuario a crear plugins de seguridad de Java:

JCCSimpleGSSPlugin.java
Una implementación de un plugin GSS-API para el servidor, que comprueba los ID de usuario y las contraseñas. Este ejemplo es una versión Java de un ejemplo del programa en lenguaje C program gssapi_simple.c.

JCCKerberosPlugin.java
Plugin de seguridad de Kerberos para el cliente. Este ejemplo es una versión Java de un ejemplo del programa en lenguaje C IBMkrb5.c.

Cuando un programa de aplicación obtiene una conexión mediante la seguridad de plugin JDBC, deberá establecer las propiedades Connection y DataSource siguientes:

Tabla 33. Configuración de la propiedad Connection o DataSource para utilizar plugins de seguridad de Java

Propiedad	Valor
com.ibm.db2.jcc.DB2BaseDataSource.user	ID de usuario utilizado para obtener la Connection.
com.ibm.db2.jcc.DB2BaseDataSource.password	Contraseña para el ID de usuario.
com.ibm.db2.jcc.DB2BaseDataSource.securityMechanism	com.ibm.db2.jcc.DB2BaseDataSource.PLUGIN_SECURITY
com.ibm.db2.jcc.DB2BaseDataSource.pluginName	Nombre del módulo de plugin para un plugin de seguridad de servidor.
com.ibm.db2.jcc.DB2BaseDataSource.plugin	Objeto del plugin para un plugin de seguridad del servidor

Ejemplo: mediante los códigos siguientes se establecen las propiedades para una conexión que utiliza la seguridad de plugin GSS-API. La conexión utiliza el plugin JCCSimpleGSSPlugin de ejemplo en el cliente y el plugin gssapi_simple de ejemplo en el servidor.

```
java.util.Properties properties = new java.util.Properties();
properties.put("user", "db2admin");
properties.put("password", "admindb2");
properties.put("pluginName", "gssapi_simple");
properties.put("securityMechanism",
  new String(""+com.ibm.db2.jcc.DB2BaseDataSource.PLUGIN_SECURITY+""));
com.ibm.db2.jcc.DB2JCCPlugin plugin =
  new com.ibm.db2.jcc.samples.plugins.JCCSimpleGSSplugin();
properties.put("plugin", plugin);
Connection con = java.sql.DriverManager.getConnection(url,
  properties);
```

Uso de mecanismos de seguridad alternativos con IBM Data Server Driver para JDBC y SQLJ

Si utiliza IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 y establece `retryWithAlternativeSecurityMechanism` en `com.ibm.db2.jcc.DB2BaseDataSource.YES` (1), y el mecanismo de seguridad original de una conexión falla, el controlador vuelve a intentar la conexión con el mecanismo de seguridad alternativo más seguro.

La tabla siguiente contiene los mecanismos de seguridad de IBM Data Server Driver para JDBC y SQLJ y los mecanismos de seguridad alternativos que se utilizan cuando la conexión original tiene un error de autorización.

Tabla 34. Mecanismos de seguridad de IBM Data Server Driver para JDBC y SQLJ original y alternativo

Tipo de autentificación del servidor	Tipo de autentificación de IBM Data Server Driver para JDBC y SQLJ para la conexión original	Tipo de autentificación de IBM Data Server Driver para JDBC y SQLJ para reintentar la conexión
CLIENT	• CLEAR_TEXT_PASSWORD_SECURITY • ENCRYPTED_PASSWORD_SECURITY • ENCRYPTED_USER_AND_PASSWORD_SECURITY • KERBEROS_SECURITY • ENCRYPTED_USER_AND_DATA_SECURITY • ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY • PLUGIN_SECURITY • ENCRYPTED_USER_ONLY_SECURITY	USER_ONLY_SECURITY
	USER_ONLY_SECURITY	Ninguno. USER_ONLY_SECURITY no falla en la conexión original.
SERVER	• USER_ONLY_SECURITY • ENCRYPTED_PASSWORD_SECURITY • ENCRYPTED_USER_AND_PASSWORD_SECURITY • KERBEROS_SECURITY • ENCRYPTED_USER_AND_DATA_SECURITY • ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY • PLUGIN_SECURITY • ENCRYPTED_USER_ONLY_SECURITY	CLEAR_TEXT_PASSWORD_SECURITY
	CLEAR_TEXT_PASSWORD_SECURITY	Ninguno. CLEAR_TEXT_PASSWORD_SECURITY no falla en la conexión original.
SERVER_ENCRYPT para DB2 Database para Linux, UNIX y Windows versión 8 Fixpack 9 o anterior	• CLEAR_TEXT_PASSWORD_SECURITY • USER_ONLY_SECURITY • KERBEROS_SECURITY • ENCRYPTED_USER_AND_DATA_SECURITY • ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY • PLUGIN_SECURITY • ENCRYPTED_USER_ONLY_SECURITY	ENCRYPTED_USER_AND_PASSWORD_SECURITY
	• ENCRYPTED_PASSWORD_SECURITY • ENCRYPTED_USER_AND_PASSWORD_SECURITY	Ninguno. ENCRYPTED_PASSWORD_SECURITY y ENCRYPTED_USER_AND_PASSWORD_SECURITY no fallan en la conexión original.
SERVER_ENCRYPT para DB2 Database para Linux, UNIX y Windows versión 8 Fixpack 10 o posterior	• USER_ONLY_SECURITY • KERBEROS_SECURITY • ENCRYPTED_USER_AND_DATA_SECURITY • ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY • PLUGIN_SECURITY • ENCRYPTED_USER_ONLY_SECURITY	ENCRYPTED_USER_AND_PASSWORD_SECURITY
	• CLEAR_TEXT_PASSWORD_SECURITY • ENCRYPTED_PASSWORD_SECURITY • ENCRYPTED_USER_AND_PASSWORD_SECURITY	Ninguno. CLEAR_TEXT_PASSWORD_SECURITY, ENCRYPTED_PASSWORD_SECURITY y ENCRYPTED_USER_AND_PASSWORD_SECURITY no fallan en la conexión original.

Tabla 34. Mecanismos de seguridad de IBM Data Server Driver para JDBC y SQLJ original y alternativo (continuación)

Tipo de autentificación del servidor	Tipo de autentificación de IBM Data Server Driver para JDBC y SQLJ para la conexión original	Tipo de autentificación de IBM Data Server Driver para JDBC y SQLJ para reintentar la conexión
DATA_ENCRYPT	• CLEAR_TEXT_PASSWORD_SECURITY • USER_ONLY_SECURITY • ENCRYPTED_PASSWORD_SECURITY • ENCRYPTED_USER_AND_PASSWORD_SECURITY • KERBEROS_SECURITY • ENCRYPTED_USER_AND_DATA_SECURITY • PLUGIN_SECURITY • ENCRYPTED_USER_ONLY_SECURITY	ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY
	ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY	Ninguno. ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY no falla en la conexión original.
KERBEROS	• CLEAR_TEXT_PASSWORD_SECURITY • USER_ONLY_SECURITY • ENCRYPTED_PASSWORD_SECURITY • ENCRYPTED_USER_AND_PASSWORD_SECURITY • ENCRYPTED_USER_AND_DATA_SECURITY • ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY • PLUGIN_SECURITY • ENCRYPTED_USER_ONLY_SECURITY	KERBEROS_SECURITY
	KERBEROS_SECURITY	Ninguno. KERBEROS_SECURITY no falla en la conexión original.
GSSPLUGIN	• CLEAR_TEXT_PASSWORD_SECURITY • USER_ONLY_SECURITY • ENCRYPTED_PASSWORD_SECURITY • ENCRYPTED_USER_AND_PASSWORD_SECURITY • KERBEROS_SECURITY • ENCRYPTED_USER_AND_DATA_SECURITY • ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY • ENCRYPTED_USER_ONLY_SECURITY	PLUGIN_SECURITY
	PLUGIN_SECURITY	Ninguno. PLUGIN_SECURITY no falla en la conexión original.
KRB_SERVER_ENCRYPT	• USER_ONLY_SECURITY • ENCRYPTED_USER_AND_DATA_SECURITY • ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY • ENCRYPTED_USER_ONLY_SECURITY	KERBEROS_SECURITY
	• CLEAR_TEXT_PASSWORD_SECURITY • ENCRYPTED_PASSWORD_SECURITY • ENCRYPTED_USER_AND_PASSWORD_SECURITY • KERBEROS_SECURITY • PLUGIN_SECURITY	Ninguno. CLEAR_TEXT_PASSWORD_SECURITY, ENCRYPTED_PASSWORD_SECURITY, ENCRYPTED_USER_AND_PASSWORD_SECURITY, KERBEROS_SECURITY y PLUGIN_SECURITY no fallan en la conexión original.
GSS_SERVER_ENCRYPT	• USER_ONLY_SECURITY • ENCRYPTED_USER_AND_DATA_SECURITY • ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY • ENCRYPTED_USER_ONLY_SECURITY	KERBEROS_SECURITY
	• CLEAR_TEXT_PASSWORD_SECURITY • ENCRYPTED_PASSWORD_SECURITY • ENCRYPTED_USER_AND_PASSWORD_SECURITY • KERBEROS_SECURITY • PLUGIN_SECURITY	Ninguno. CLEAR_TEXT_PASSWORD_SECURITY, ENCRYPTED_PASSWORD_SECURITY, ENCRYPTED_USER_AND_PASSWORD_SECURITY, KERBEROS_SECURITY y PLUGIN_SECURITY no fallan en la conexión original.

Soporte a contextos fiables del controlador de IBM Data Server para JDBC y SQLJ

IBM Data Server Driver para JDBC y SQLJ proporciona métodos que permiten establecer y utilizar conexiones fiables en programas Java.

Las conexiones fiables están soportadas para:

- IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 para:
  - DB2 Database para Linux, UNIX y Windows Versión 9.5 o versiones posteriores
  - DB2 para z/OS Versión 9.1 o versiones posteriores
  - IBM Informix Versión 11.70 o versiones posteriores
- IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS Versión 9.1 o versiones posteriores

Un modelo de aplicación de tres niveles consta de un servidor de bases de datos, un servidor de middleware, tal como WebSphere Application Server, y usuarios finales. Con este modelo, el servidor de middleware tiene la función de acceder al servidor de bases de datos en nombre de los usuarios finales. El soporte de contexto fiable garantiza la utilización de los privilegios de base de datos y de la identidad de base de datos del usuario final cuando se servidor de middleware realiza peticiones de base de datos en nombre del usuario final.

Un contexto fiable es un objeto que el administrador de la base de datos define y que contiene un ID de autorización del sistema y un conjunto de atributos de confianza. Actualmente, para los servidores de bases de datos DB2, una conexión de base de datos es el único tipo de contexto que se puede utilizar. Los atributos de confianza identifican un conjunto de características de una conexión que son necesarias para que la conexión se considere fiable. La relación entre una conexión de base de datos y un contexto fiable se establece cuando se crea por primera vez la conexión con el servidor de bases de datos. Esa relación persiste durante todo el tiempo de vida de la conexión de base de datos.

Después de definir un contexto fiable y establecer una conexión inicial fiable con el servidor de bases de datos, el servidor de middleware puede utilizar esa conexión de base de datos bajo un usuario diferente sin volver a autentificar al nuevo usuario en el servidor de bases de datos.

Para evitar la vulnerabilidad a errores de seguridad, un servidor de aplicaciones que utilice estos métodos fiables no debe utilizar métodos de conexión no fiables.

La clase DB2ConnectionPoolDataSource proporciona varias versiones del método getDB2TrustedPooledConnection, y la clase DB2XADataSource proporciona varias versiones del método getDB2TrustedXAConnection, lo que permite que un servidor de aplicaciones establezca una conexión inicial fiable. Seleccione un método de acuerdo con los tipos de propiedades de conexión que pasará al programa y si se utilizará la seguridad Kerberos. Cuando un servidor de aplicaciones invoca uno de estos métodos, IBM Data Server Driver para JDBC y SQLJ devuelve una matriz Object[] con dos elementos:

- El primer elemento contiene una instancia de conexión para la conexión inicial.
- El segundo elemento contiene una cookie exclusiva para la instancia de conexión. La cookie es generada por el controlador JDBC y se utiliza para la autentificación durante la reutilización posterior de la conexión.

La clase DB2PooledConnection proporciona varias versiones del método getDB2Connection, y la clase DB2Connection proporciona varias versiones del método reuseDB2Connection. Esto permite que un servidor de aplicaciones pueda reutilizar una conexión fiable existente a petición de un usuario nuevo. El servidor de aplicaciones utiliza el método para pasar los elementos siguientes al usuario nuevo:

- La cookie de la conexión inicial
- Nuevas propiedades de conexión para la conexión reutilizada

El controlador JDBC comprueba que la cookie proporcionada coincida con la cookie de la conexión física fiable subyacente con el fin de asegurar que la petición de conexión proviene del servidor de aplicaciones que estableció la conexión física fiable. Si la cookie coincide, el nuevo usuario podrá utilizar de inmediato la conexión con las nuevas propiedades.

Ejemplo: obtención de la conexión fiable inicial:

```
// Crear una instancia DB2ConnectionPoolDataSource
com.ibm.db2.jcc.DB2ConnectionPoolDataSource dataSource =
  new com.ibm.db2.jcc.DB2ConnectionPoolDataSource();
// Definir propiedades para esta instancia
dataSource.setDatabaseName ("STLEC1");
dataSource.setServerName ("v7ec167.svl.ibm.com");
dataSource.setDriverType (4);
dataSource.setPortNumber(446);
java.util.Properties properties = new java.util.Properties();
// Definir otras propiedades mediante
// properties.put("propiedad", "valor");
// Proporcionar el ID de usuario y contraseña para la conexión
String user = "user";
String password = "password";
// Invocar getDB2TrustedPooledConnection para obtener una
// instancia de conexión fiable y la cookie para la conexión
Object[] objects = dataSource.getDB2TrustedPooledConnection(
                        user,password, properties);
```

Ejemplo: Reutilización de una conexión fiable existente:

```
// Primer elemento obtenido de llamada anterior a getDB2TrustedPooledConnection
// es un objeto de conexión. Convertirlo en un objeto PooledConnection.
javax.sql.PooledConnection pooledCon =
  (javax.sql.PooledConnection)objects[0];
properties = new java.util.Properties();
// Definir propiedades nuevas para el objeto reutilizado mediante
// properties.put("propiedad", "valor");
// El segundo elemento obtenido de llamada anterior a getDB2TrustedPooledConnection
// es la cookie de la conexión. Convertirla como matriz de bytes.
byte[] cookie = ((byte[])(objects[1]);
// Proporcionar el ID de usuario para la conexión nueva.
String newuser = "newuser";
// Suministrar el nombre de un servicio de correlación que correlacione
// el ID de usuario de estación de una estación de trabajo
// con un ID de z/OS RACF
String userRegistry = "registry";
// No proporcionar ningún símbolo de certificado de seguridad que deba rastrearse.
byte[] userSecTkn = null;
// No proporcionar ningún ID de usuario anterior.
String originalUser = null;
// Invocar getDB2Connection para obtener un objeto de conexión para el
// usuario nuevo.
java.sql.Connection con =
  ((com.ibm.db2.jcc.DB2PooledConnection)pooledCon).getDB2Connection(
     cookie,newuser,password,userRegistry,userSecTkn,originalUser,properties);
```

Soporte para SSL de IBM Data Server Driver para JDBC y SQLJ

El controlador IBM Data Server Driver para JDBC y SQLJ permite utilizar Secure Sockets Layer (SSL) mediante Java Secure Socket Extension (JSSE).

Puede utilizar SSL en sus aplicaciones Java si utiliza IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 con DB2 para z/OS Versión 9 o posterior, con DB2 Database para Linux, UNIX y Windows Versión 9.1, Fixpack 2 o posterior, o con IBM Informix Versión 11.50 o posterior.

Si se utiliza el soporte de SSL para una conexión con un servidor de datos de DB2 para z/OS y la versión de z/OS es V1.8, V1.9 o V1.10, el PTF apropiado para el APAR PK72201 debe aplicarse a Communication Server para los servicios IP de z/OS.

Las conexiones con todos los servidores de datos soportados pueden utilizar la autentificación de servidor. Para la autentificación de servidor, el servidor envía un certificado al cliente y el cliente confirma la identidad del servidor. Las conexiones con los servidores de datos de DB2 para z/OS pueden utilizar también la autentificación de cliente. Para la autentificación de cliente, el cliente envía un certificado al servidor y el servidor confirma la identidad del cliente. La autentificación de cliente se puede utilizar con cifrado SSL o sin cifrado SSL.

Para utilizar conexiones SSL, debe realizar lo siguiente:

- Configurar conexiones con el servidor de datos para utilizar SSL.
- Configurar el Entorno de ejecución Java para que utilice SSL.

Configuración de conexiones en IBM Data Server Driver para JDBC y SQLJ para que utilicen SSL

Para configurar conexiones de bases de datos en IBM Data Server Driver para JDBC y SQLJ para que utilicen SSL, es preciso que establezca la propiedad DB2BaseDataSource.sslConnection en `true`.

Antes de empezar

Para que una conexión a una fuente de datos pueda utilizar SSL, el puerto al que se conecta la aplicación debe configurarse en el servidor de bases de datos como el puerto de escucha SSL.

Procedimiento

1. Establezca DB2BaseDataSource.sslConnection en una instancia Connection o DataSource.
2. Opcional: Establezca DB2BaseDataSource.sslTrustStoreLocation en una instancia Connection o DataSource para identificar la ubicación del almacén de confianza. Establecer la propiedad sslTrustStoreLocation es una alternativa a establecer la propiedad de Java javax.net.ssl.trustStore. Si establece DB2BaseDataSource.sslTrustStoreLocation, javax.net.ssl.trustStore no se utiliza.
3. Opcional: Establezca DB2BaseDataSource.sslTrustStorePassword en una instancia Connection o DataSource para identificar la contraseña del almacén de confianza. Establecer la propiedad sslTrustStorePassword es una alternativa a establecer la propiedad de Java javax.net.ssl.trustStorePassword. Si establece DB2BaseDataSource.sslTrustStorePassword, javax.net.ssl.trustStorePassword no se utiliza.

Ejemplo

El ejemplo siguiente muestra cómo establecer la propiedad sslConnection en una instancia Connection:

```
java.util.Properties properties = new java.util.Properties();
properties.put("user", "xxxx");
properties.put("password", "yyyy");
properties.put("sslConnection", "true");
java.sql.Connection con =
  java.sql.DriverManager.getConnection(url, properties);
```

Configuración del Entorno de ejecución Java para que utilice SSL

Para poder utilizar conexiones SSL (Secure Sockets Layer) en las aplicaciones JDBC y SQLJ, es necesario que configure el entorno de ejecución Java para que utilice SSL. Se proporciona un procedimiento de ejemplo. Sin embargo, el procedimiento puede ser diferente dependiendo del entorno de ejecución Java que utilice.

Antes de empezar

Para poder ejecutar el entorno de ejecución Java para SSL, es necesario que cumpla los requisitos previos siguientes:

- El entorno de ejecución Java debe incluir un proveedor de seguridad Java. Debe instalarse el proveedor IBM JSSE o el proveedor SunJSSE. El proveedor IBM JSSE se instala automáticamente con el SDK de IBM para Java.

  Restricción: Sólo se puede utilizar el proveedor SunJSSE con un entorno de ejecución Java de Oracle. El proveedor SunJSSE no funciona con un entorno de ejecución Java de IBM.
- El soporte de SSL debe configurarse en el servidor de bases de datos.

Acerca de esta tarea

Para configurar el entorno de ejecución Java para que utilice SSL, realice los pasos siguientes.

Procedimiento

1. Importe un certificado del servidor de bases de datos al almacén de confianza de Java en el cliente.

   Utilice el programa de utilidad Java keytool para importar el certificado al almacén de confianza.

   Por ejemplo, suponga que el certificado del servidor está almacenado en un archivo denominado jcc.cacert. Emita la siguiente sentencia del programa de utilidad keytool para leer el certificado del archivo jcc.cacert y almacenarlo en un almacén de confianza denominado cacerts.

   ```
   keytool -import -file jcc.cacert -keystore cacerts
   ```
2. Configure el entorno de ejecución Java para los proveedores de seguridad Java añadiendo entradas al archivo java.security.

   El formato de una entrada de proveedor de seguridad es:

   ```
   security.provider.n=nombre-paquete-proveedor
   ```

   Un proveedor con un valor inferior a *n* tiene prioridad sobre un proveedor con un valor superior a *n*.

Las entradas del proveedor de seguridad Java que añada dependen de si utiliza el proveedor JSSE de IBM o el proveedor SunJSSE.

- Si utiliza el proveedor SunJSSE, añada entradas para los proveedores de seguridad Oracle al archivo java.security.
- Si utiliza el proveedor IBM JSSE, utilice uno de los métodos siguientes:
  - **Utilice el proveedor IBMJSSE2 (soportado para el SDK de IBM para Java 1.4.2 y superior:**

    Recomendación: utilice el proveedor IBMJSSE2 y utilícelo en la modalidad FIPS.
    - Si no necesita operar en modalidad compatible con FIPS:
      - Para el SDK de IBM para Java 1.4.2, añada una entrada para IBMJSSE2Provider al archivo java.security. Asegúrese de que haya una entrada para el proveedor IBMJCE en el archivo java.security. El archivo de seguridad que se suministra con el SDK de IBM para Java contiene una entrada para entradas de IBMJCE.
      - Para versiones posteriores del SDK de IBM para Java, asegúrese de que las entradas del proveedor IBMJSSE2Provider y IBMJCE se encuentren en el archivo java.security. El archivo java.security que se ha entregado con el SDK de IBM para Java contiene entradas para estos proveedores.
    - Si necesita operar en modalidad compatible con FIPS:
      - Añada una entrada para el proveedor IBMJCEFIPS en el archivo java.security antes de la entrada para el proveedor IBMJCE. No elimine la entrada del proveedor IBMJCE.
      - Habilite la modalidad FIPS en el proveedor IBMJSSE2. Consulte el paso 3 en la página 230.
  - **Utilice el proveedor IBMJSSE (soportado únicamente para el SDK de IBM para Java 1.4.2):**
    - Si no necesita operar en modalidad compatible con FIPS, asegúrese de que las entradas del proveedor IBMJSSEProvider y IBMJCE se encuentren en el archivo java.security. El archivo java.security que se ha entregado con el SDK de IBM para Java contiene entradas para estos proveedores.
    - Si necesita operar en modalidad compatible con FIPS, añada entradas para el proveedor aprobado por FIPS IBMJSSEFIPSProvider y el proveedor IBMJCEFIPS en el archivo java.security, antes de la entrada del proveedor IBMJCE.

  Restricción: Si utiliza el proveedor IBMJSSE en el sistema operativo Solaris, necesita incluir una entrada para el proveedor SunJSSE antes de las entradas de los proveedores IBMJCE, IBMJCEFIPS, IBMJSSE o IBMJSSE2.

Ejemplo: Utilice un archivo java.security similar a éste si necesita ejecutar en modalidad compatible con FIPS y habilite la modalidad FIPS en el proveedor IBMJSSE2:

```
# Establecer los proveedores de seguridad de Java
security.provider.1=com.ibm.jsse2.IBMJSSEProvider2
security.provider.2=com.ibm.crypto.fips.provider.IBMJCEFIPS
security.provider.3=com.ibm.crypto.provider.IBMJCE
security.provider.4=com.ibm.security.jgss.IBMJGSSProvider
security.provider.5=com.ibm.security.cert.IBMCertPath
security.provider.6=com.ibm.security.sasl.IBMSASL
```

Ejemplo: Utilice un archivo java.security similar a éste si necesita ejecutar en modalidad compatible con FIPS y utilice el proveedor IBMJSSE:

```
# Establecer los proveedores de seguridad de Java
security.provider.1=com.ibm.fips.jsse.IBMJSSEFIPSProvider
security.provider.2=com.ibm.crypto.fips.provider.IBMJCEFIPS
security.provider.3=com.ibm.crypto.provider.IBMJCE
security.provider.4=com.ibm.security.jgss.IBMJGSSProvider
security.provider.5=com.ibm.security.cert.IBMCertPath
security.provider.6=com.ibm.security.sasl.IBMSASL
```

Ejemplo: Utilice un archivo java.security similar a éste si utiliza el proveedor SunJSSE:

```
# Establecer los proveedores de seguridad de Java
security.provider.1=sun.security.provider.Sun
security.provider.2=com.sun.rsajca.Provider
security.provider.3=com.sun.crypto.provider.SunJCE
security.provider.4=com.sun.net.ssl.internal.ssl.Provider
```

3. Si tiene la intención de utilizar IBM Data Server Driver para JDBC y SQLJ en modalidad compatible con FIPS, necesita establecer la propiedad del sistema Java `com.ibm.jsse2.JSSEFIPS`:

   ```
   com.ibm.jsse2.JSSEFIPS=true
   ```

   Restricción: Las aplicaciones JSSE que no estén de modalidad FIPS no se pueden ejecutar en una JVM que esté en modalidad FIPS.

   Restricción: Cuando el proveedor IBMJSSE2 se ejecuta en modalidad FIPS, no puede utilizar la criptografía por hardware.

4. Configure el entorno de ejecución Java para los proveedores de fábrica de sockets SSL añadiendo entradas al archivo java.security. Este paso no es necesario si está utilizando el proveedor SunJSSE y el entorno de ejecución Java, 7 o posterior.

   El formato de las entradas del proveedor de fábrica de sockets SSL es:

   ```
   ssl.SocketFactory.provider=nombre-paquete-proveedor
   ssl.ServerSocketFactory.provider=nombre-paquete-proveedor
   ```

   Especifique el proveedor de fábrica de sockets SSL para el proveedor de seguridad Java que está utilizando.

   Ejemplo: Incluya entradas del proveedor de fábrica de sockets SSL como éstas en el archivo java.security cuando habilite la modalidad FIPS en el proveedor IBMJSSE2:

   ```
   # Establecer el proveedor de fábrica de sockets SSL
   ssl.SocketFactory.provider=com.ibm.jsse2.SSLSocketFactoryImpl
   ssl.ServerSocketFactory.provider=com.ibm.jsse2.SSLServerSocketFactoryImpl
   ```

   Ejemplo: Incluya entradas del proveedor de fábrica de sockets SSL como éstas en el archivo java.security cuando habilite la modalidad FIPS en el proveedor IBMJSSE:

   ```
   # Establecer el proveedor de fábrica de sockets SSL
   ssl.SocketFactory.provider=com.ibm.fips.jsse.JSSESocketFactory
   ssl.ServerSocketFactory.provider=com.ibm.fips.jsse.JSSEServerSocketFactory
   ```

   Ejemplo: Incluya entradas del proveedor de fábrica de sockets SSL como éstas cuando utilice el proveedor SunJSSE y el entorno de ejecución Java, 6 o anterior:

   ```
   # Establecer el proveedor de fábrica de sockets SSL
   ssl.SocketFactory.provider=com.sun.net.ssl.internal.ssl.SSLSocketFactoryImpl
   ssl.ServerSocketFactory.provider=com.sun.net.ssl.internal.ssl.SSLServerSocketFactoryImpl
   ```

5. Configure las propiedades del sistema Java para que utilicen el almacén de confianza.

   Para ello, establezca las siguientes propiedades del sistema Java:

javax.net.ssl.trustStore
Especifica el nombre del almacén de confianza que ha especificado con el parámetro -keystore en el programa de utilidad keytool del paso 1 en la página 228.

Si la propiedad de IBM Data Server Driver para JDBC y SQLJ DB2BaseDataSource.sslTrustStoreLocation está establecida, su valor prevalece sobre el valor de la propiedad javax.net.ssl.trustStore.

javax.net.ssl.trustStorePassword (opcional)
Especifica la contraseña para el almacén de confianza. No es necesario establecer una contraseña del almacén de confianza. No obstante, si no establece la contraseña, no puede proteger la integridad del almacén de confianza.

Si la propiedad de IBM Data Server Driver para JDBC y SQLJ DB2BaseDataSource.sslTrustStorePassword está establecida, su valor prevalece sobre el valor de la propiedad javax.net.ssl.trustStorePassword.

Ejemplo: uno de que pueda establecer las propiedades del sistema Java es especificarlas como los argumentos de la opción -D cuando ejecute una aplicación Java. Suponga que desea ejecutar una aplicación Java denominada MySSL.java, que accede a una fuente de datos utilizando una conexión SSL. Ha definido un almacén de confianza denominado cacerts. El mandato siguiente establece el nombre del almacén de confianza cuando ejecuta la aplicación.

```
java -Djavax.net.ssl.trustStore=cacerts MySSL
```

Soporte de IBM Data Server Driver para JDBC y SQLJ para la autentificación de certificados

IBM Data Server Driver para JDBC y SQLJ proporciona soporte para el soporte de cliente para la autentificación de certificados para las conexiones con servidores de datos DB2 para z/OS Versión 10 o posteriores.

La seguridad de autentificación de certificado de cliente en un servidor de datos DB2 para z/OS soporta el uso de certificados digitales para la autentificación mutua por peticionarios y servidores. Al utilizar certificados digitales de z/OS, el protocolo SSL (Secure Socket Layer) da soporte a la autentificación de servidor y de cliente durante la fase de reconocimiento. Un servidor de datos puede validar los certificados de un cliente en el servidor, que impide que el cliente obtenga una conexión segura sin un certificado aprobado por la instalación. La autentificación del certificado digital del cliente remoto se realiza mediante el protocolo AT-TLS (Application Transparent Transport Layer Security) que se proporciona con la pila z/OS Communications Server TCP/IP.

IBM Data Server Driver para JDBC y SQLJ soporta la autentificación de certificados para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 únicamente.

Puede habilitar la autentificación de certificados de IBM Data Server Driver para JDBC y SQLJ especificando DB2BaseDataSource.TLS_CLIENT_CERTIFICATE_SECURITY como el valor de la propiedad Connection o DataSource de securityMechanism. Si el servidor de datos de destino soporta la autentificación de certificados de cliente y la autentificación mutua se ejecuta satisfactoriamente, el controlador pasa un objeto Connection válido a la aplicación. Si el servidor de datos no da soporte a autentificación de

certificados de cliente, o la conexión no se ha autentificado mediante AT-TLS y SSL, el controlador emite una DisconnectException.

Puede utilizar la autentificación de certificados con o sin un ID de usuario o una contraseña. Si la aplicación no proporciona un ID de usuario o una contraseña, la autentificación se realiza en la capa de red únicamente. Si se proporciona un ID de usuario o una contraseña, la autentificación se realiza en la capa de red y en la capa del servidor de datos.

Para utilizar el cifrado SSL con la autentificación de certificados, puede establecer la propiedad de Connection o DataSource de sslConnection en `true`.

El ejemplo siguiente muestra cómo habilitar la autentificación de certificados y la seguridad por ID de usuario y contraseña en una aplicación JDBC.

```
com.ibm.db2.jcc.DB2SimpleDataSource dataSource = new
  com.ibm.db2.jcc.DB2SimpleDataSource();
// Especifique la autentificación de certificado
dataSource.setSecurityMechanism
(com.ibm.db2.jcc.DB2BaseDataSource.TLS_CLIENT_CERTIFICATE_SECURITY);
// Establezca el ID de usuario y la contraseña que se pasarán al servidor de datos
((com.ibm.db2.jcc.DB2BaseDataSource)dataSource).setUser("sysadm");
dataSource.setPassword("password");
// Identifique el almacén de confianza de SSL, el almacén de
// claves y sus contraseñas
System.setProperty("javax.net.ssl.trustStore","c:/temp/SSL/cacerts");
System.setProperty("javax.net.ssl.trustStorePassword","password");
System.setProperty("javax.net.ssl.keyStore","c:/temp/SSL/myKS");
System.setProperty("javax.net.ssl.keyStorePassword","123456");
...
// Cree una conexión
con = dataSource.getConnection ();
```

Seguridad para preparar aplicaciones SQLJ con IBM Data Server Driver para JDBC y SQLJ

Puede proporcionar seguridad durante la preparación de aplicaciones SQLJ permitiendo a los usuarios personalizar las aplicaciones solamente y limitando el acceso a un determinado conjunto de tablas durante la personalización. Si el servidor de datos de destino es DB2 para z/OS, también puede proporcionar seguridad permitiendo a los clientes que preparen las aplicaciones, pero sin ejecutarlas.

Permitir a los usuarios sólo la personalización

Puede utilizar una de las técnicas siguientes para permitir a un conjunto de usuarios personalizar las aplicaciones SQLJ, pero no vincular ni ejecutar esas aplicaciones:

- **Crear un sistema de bases de datos para la personalización solamente (solución recomendada):** Siga estos pasos:
  1. Cree una nueva instancia del gestor de bases de datos. Éste es el sistema de sólo personalización.
  2. En el sistema de sólo personalización, defina todas las tablas y vistas a las que acceden las aplicaciones SQLJ. Las definiciones de tablas o vistas deben ser las mismas definiciones existentes en la instancia del gestor de bases de datos donde se vinculará y ejecutará la aplicación (el sistema de vinculación

y ejecución). La ejecución de la sentencia DESCRIBE en las tablas o vistas debe dar los mismos resultados en el sistema de sólo personalización y en el sistema de vinculación y ejecución.

3. En el sistema de sólo personalización, otorgue los privilegios necesarios de vista o tabla a los usuarios que personalizarán las aplicaciones SQLJ.
4. En el sistema de sólo personalización, los usuarios ejecutan el mandato sqlj con la opción -compile=true para crear perfiles serializados y códigos de bytes de Java para sus programas. A continuación ejecutan el mandato db2sqljcustomize con la opción -automaticbind NO para crear los perfiles serializados personalizados.
5. Copie los perfiles serializados personalizados y los archivos de código de bytes de java en el sistema de vinculación y ejecución.
6. Un usuario con autorización para vincular paquetes en el sistema de vinculación y ejecución ejecuta el mandato db2sqljbind en los perfiles serializados personalizados que se han copiado desde el sistema de sólo personalización.

- **Utilice un procedimiento almacenado para la personalización:** Grabe un procedimiento almacenado de Java que personalice los perfiles serializados y vincule paquetes para las aplicaciones SQLJ en beneficio del usuario final. Este procedimiento almacenado de Java necesita el uso de un paquete del controlador JDBC que se ha vinculado con una de las opciones DYNAMICRULES que provoca la ejecución de SQL dinámico bajo un ID de usuario diferente desde el ID de autorización del usuario final. Por ejemplo, puede utilizar la opción DYNAMICRULES con DEFINEBIND o DEFINERUN para ejecutar SQL dinámico bajo el ID de autorización del creador del procedimiento almacenado de Java. Debe otorgar la autorización EXECUTE en el procedimiento almacenado a los usuarios que necesiten realizar la personalización de SQLJ.

  El procedimiento almacenado hace lo siguiente:

  1. Recibe el programa SQLJ compilado y los perfiles serializados en los parámetros de entrada BLOB
  2. Copia los parámetros de entrada en su sistema de archivos
  3. Ejecuta db2sqljcustomize para personalizar los perfiles serializados y vincular los paquetes para el programa SQLJ
  4. Devuelve los perfiles serializados personalizados en los parámetros de salida
- **Utilice un programa autónomo para la personalización:** Esta técnica implica la grabación de un programa que ejecuta los mismos pasos que un procedimiento almacenado de Java, que personaliza los perfiles serializados y vincula paquetes para las aplicaciones SQLJ en beneficio del usuario final. Sin embargo, en lugar de ejecutar el programa como procedimiento almacenado, ejecuta el programa como autónomo, bajo un servidor de bibliotecas.

Permitir a los usuarios sólo la personalización y vinculación

Si el servidor de datos de destino es DB2 para z/OS Versión 10 o posterior, puede permitir a los usuarios personalizar y vincular las aplicaciones SQLJ pero no ejecutar las sentencias de SQL que contienen, otorgando el privilegio EXPLAIN a aquellos usuarios.

Restricción del acceso a las tablas durante la personalización

Cuando personalice perfiles serializados, debe efectuar comprobaciones en línea, para proporcionar al programa de aplicación información sobre los tipos de datos y

longitudes de las columnas de las tablas a las que accede el programa. Por omisión, la personalización incluye la comprobación en línea.

La comprobación en línea requiere que el usuario que personaliza un perfil serializado tenga autorización para ejecutar sentencias PREPARE y DESCRIBE con sentencias de SQL en el programa SQLJ. Esa autorización incluye el privilegio SELECT en las tablas y vistas a las que acceden las sentencias de SQL. Si las sentencias de SQL contienen nombres de tablas sin calificar, el calificador que se utiliza durante la comprobación en línea es el valor del parámetro db2sqljcustomize -qualifier. Por consiguiente, para la comprobación en línea de tablas y vistas con nombres sin calificar en una aplicación SQLJ, sólo puede otorgar el privilegio SELECT en las tablas y vistas con un calificador que coincida con el valor del parámetro -qualifier.

Capítulo 6. Creación de aplicaciones de bases de datos Java

Puede crear manualmente aplicaciones de base de datos para JDBC y SQLJ. Como alternativa, puede utilizar un makefile Java para crear aplicaciones JDBC y utilizar el archivo bldsqlj que se proporciona con DB2 Database para Linux, UNIX y Windows para crear aplicaciones SQLJ.

Creación de applets JDBC

Puede utilizar un archivo `makefile` de Java o ejecutar manualmente el mandato javac para crear aplicaciones JDBC.

Acerca de esta tarea

Los pasos siguientes muestran cómo crear y ejecutar el applet de ejemplo `Applt.java` de JDBC.

Procedimiento

1. Compile `Applt.java` para obtener el archivo `Applt.class` mediante este mandato:

   ```
   javac Applt.java
   ```

2. Compruebe que su navegador Web o visor de applets Java (si lo utiliza) pueda acceder a su directorio de trabajo. Si su directorio no es accesible, copie los archivos siguientes en un directorio que sea accesible:
   - Applt.html
   - Applt.class
3. Copie `sqllib\java\db2jcc.jar` en Windows o `sqllib/java/db2jcc.jar` en UNIX, en el mismo directorio que `Applt.class` y `Applt.html`.

   Si utiliza cualquiera de las funciones de JDBC 4.0 o posteriores, copie db2jcc4.jar en lugar de db2jcc.jar.
4. Si está utilizando IBM Data Server Driver para JDBC y SQLJ, conecte con ese controlador modificando el archivo `Applt.html` de acuerdo con las instrucciones contenidas en el archivo. Para el número de puerto TCP/IP, debe utilizar el número de puerto 50000 de la base de datos.
5. Para ejecutar este applet, compruebe que esté instalado y en ejecución un servidor Web en la máquina DB2 (servidor o cliente), o que puede utilizar el visor de applets proporcionado con el SDK de Java. Para ello emita el mandato siguiente desde el directorio de trabajo de la máquina cliente:

   ```
   appletviewer Applt.html
   ```

Creación de aplicaciones JDBC

Puede utilizar un archivo `makefile` de Java o ejecutar manualmente el mandato javac para crear aplicaciones JDBC.

Acerca de esta tarea

Los pasos siguientes muestran cómo crear y ejecutar la aplicación SQLJ de ejemplo `DbInfo` de JDBC.

Procedimiento

1. Compile `DbInfo.java` para obtener el archivo `DbInfo.class` mediante este mandato:

   ```
   javac DbInfo.java
   ```

2. Si está ejecutando una aplicación Java de UNIX en una instancia de DB2 de 64 bits, pero el SDK (software development kit) de Java es de 32 bits, es necesario que cambie la vía de acceso de la biblioteca de DB2 antes de ejecutar la aplicación. Por ejemplo, en AIX:
   - En el shell bash o Korn:

     ```
     export LIBPATH=$HOME/sqllib/lib32
     ```

   - En el shell C:

     ```
     setenv LIBPATH $HOME/sqllib/lib32
     ```

3. Ejecute el intérprete de Java en la aplicación con este mandato:

   ```
   java DbInfo
   ```

Creación de rutinas JDBC

Puede utilizar un archivo `makefile` de Java o el mandato javac para crear rutinas JDBC. Una vez creadas esas rutinas, es necesario catalogarlas.

Acerca de esta tarea

Los pasos siguientes muestran cómo crear y ejecutar estas rutinas:

- El procedimiento almacenado de ejemplo `SpServer` de JDBC
- La función definida por el usuario de ejemplo `UDFsrv`, que no tiene ninguna sentencia de SQL
- La función definida por el usuario de ejemplo `UDFsqlsv`, que tiene sentencias de SQL

Procedimiento

- Para crear y ejecutar el procedimiento almacenado SpServer.java en el servidor, desde la línea de mandatos:
  1. Compile SpServer.java para crear el archivo SpServer.class, mediante este mandato:

     ```
     javac SpServer.java
     ```

  2. Copie `SpServer.class` en el directorio `sqllib\function` (en sistemas operativos Windows) o en el directorio `sqllib/function` (en UNIX).
  3. Catalogue las rutinas ejecutando el script spcat en el servidor. El script spcat conecta con la base de datos de ejemplo, descataloga las rutinas si fueron catalogadas previamente mediante SpDrop.db2, luego las cataloga utilizando SpCreate.db2, y finalmente desconecta de la base de datos. También puede ejecutar los scripts SpDrop.db2 y SpCreate.db2 por separado.
  4. Detenga y reinicie la base de datos para que se pueda reconocer el nuevo archivo de clase. Si es necesario, defina la modalidad de archivo del archivo de clase como "read" para que pueda ser leído por el usuario delimitado.
  5. Compile y ejecute la aplicación cliente `SpClient` para acceder a la clase del procedimiento almacenado.
- Para crear y ejecutar el programa de la función definida por el usuario `UDFsrv.java` (función definida por el usuario sin ninguna sentencia de SQL) en el servidor, realice lo siguiente desde la línea de mandatos:

1. Compile `UDFsrv.java` para crear el archivo UDFsrv.class, mediante este mandato:

   ```
   javac UDFsrv.java
   ```

2. Copie `UDFsrv.class` en el directorio `sqllib\function` (en sistemas operativos Windows) o en el directorio `sqllib/function` (en UNIX).
3. Compile y ejecute un programa cliente por el que se invoque UDFsrv.

   Para acceder a la biblioteca de `UDFsrv`, puede utilizar la aplicación UDFcli.java de JDBC o la aplicación cliente UDFcli.sqlj de SQLJ. Ambas versiones del programa cliente contienen la sentencia CREATE FUNCTION de SQL, que permite registrar las funciones definidas por el usuario en la base de datos, y también contiene sentencias de SQL que hacen uso de funciones definidas por el usuario.

- Para crear y ejecutar el programa de la función definida por el usuario `UDFsqlsv.java` (función definida por el usuario con sentencias de SQL) en el servidor, realice lo siguiente desde la línea de mandatos:
  1. Compile `UDFsqlsv.java` para crear el archivo `UDFsqlsv.class`, mediante este mandato:

     ```
     javac UDFsqlsv.java
     ```

  2. Copie `UDFsqlsv.class` en el directorio `sqllib\function` (en sistemas operativos Windows) o en el directorio `sqllib/function` (en UNIX).
  3. Compile y ejecute un programa cliente por el que se invoque UDFsqlsv.

     Para acceder a la biblioteca de `UDFsqlsv`, puede utilizar la aplicación `UDFsqlcl.java` de JDBC. El programa cliente contiene la sentencia CREATE FUNCTION de SQL, que permite registrar las funciones definidas por el usuario en la base de datos, y también contiene sentencias de SQL que hacen uso de funciones definidas por el usuario.

Creación de applets SQLJ

Puede utilizar un archivo `makefile` de Java o el archivo de creación `bldsqlj` para crear applets SQLJ.

Acerca de esta tarea

Los pasos siguientes muestran cómo crear y ejecutar el applet SQLJ de ejemplo `Applt`. Estos pasos utilizan el archivo de creación `bldsqlj` (UNIX), o `bldsqlj.bat` (Windows), el cual contiene mandatos para crear un applet o aplicación SQLJ.

El archivo de creación utiliza como entrada un máximo de seis parámetros: $1, $2, $3, $4, $5 y $6 en UNIX y %1, %2, %3, %4, %5 y %6 en Windows. El primer parámetro especifica el nombre del programa. El segundo parámetro especifica el ID de usuario correspondiente a la instancia de base de datos; el tercer parámetro especifica la contraseña. El cuarto parámetro especifica el nombre del servidor. El quinto parámetro especifica el número de puerto. Finalmente, el sexto parámetro especifica el nombre de la base de datos. Se pueden utilizar valores por omisión para todos los parámetros, excepto para el primero, el nombre del programa. Consulte el archivo de creación para conocer detalles sobre la utilización de valores de parámetro por omisión.

Procedimiento

1. Cree el applet, utilizando este mandato:

   ```
   bldsqlj Applt <ID_usuario> <contraseña> <nombre_servidor> <número_puerto>
                         <nombre_base_datos>
   ```

2. Compruebe que su navegador Web o visor de applets Java (si lo utiliza) pueda acceder a su directorio de trabajo. Si su directorio no es accesible, copie los archivos siguientes en un directorio que sea accesible:
   - Applt.html
   - Applt.class
   - Applt_Cursor1.class
   - Applt_Cursor2.class
   - Applt_SJProfileKeys.class
   - Applt_SJProfile0.ser
3. Copie `sqllib\java\db2jcc.jar` en Windows o `sqllib/java/db2jcc.jar` en UNIX, en el mismo directorio que `Applt.class` y `Applt.html`.

   Si utiliza cualquiera de las funciones de JDBC 4.0 o posteriores, copie db2jcc4.jar en lugar de db2jcc.jar.
4. Si está utilizando IBM Data Server Driver para JDBC y SQLJ, conecte con ese controlador modificando el archivo `Applt.html` de acuerdo con las instrucciones contenidas en el archivo. Para el número de puerto TCP/IP, debe utilizar el número de puerto 50000 de la base de datos.
5. Para ejecutar este applet, compruebe que esté instalado y en ejecución un servidor Web en la máquina DB2 (servidor o cliente), o que puede utilizar el visor de applets proporcionado con el SDK de Java. Para ello emita el mandato siguiente desde el directorio de trabajo de la máquina cliente:

   ```
   appletviewer Applt.html
   ```

Creación de aplicaciones SQLJ

Puede utilizar un archivo `makefile` de Java o el archivo de creación `bldsqlj` para crear aplicaciones SQLJ.

Acerca de esta tarea

Los pasos siguientes muestran cómo crear y ejecutar la aplicación SQLJ de ejemplo `TbMod`. Estos pasos utilizan el archivo de creación `bldsqlj` (UNIX), o `bldsqlj.bat` (Windows), el cual contiene mandatos para crear un applet o aplicación SQLJ.

El archivo de creación utiliza como entrada un máximo de seis parámetros: $1, $2, $3, $4, $5 y $6 en UNIX y %1, %2, %3, %4, %5 y %6 en Windows. El primer parámetro especifica el nombre del programa. El segundo parámetro especifica el ID de usuario correspondiente a la instancia de base de datos; el tercer parámetro especifica la contraseña. El cuarto parámetro especifica el nombre del servidor. El quinto parámetro especifica el número de puerto. Finalmente, el sexto parámetro especifica el nombre de la base de datos. Se pueden utilizar valores por omisión para todos los parámetros, excepto para el primero, el nombre del programa. Consulte el archivo de creación para conocer detalles sobre la utilización de valores de parámetro por omisión.

Procedimiento

1. Cree la aplicación con este mandato:

   ```
   bldsqlj TbMod <ID de usuario> <contraseña> <nombre_servidor> <número_puerto>
                        <nombre_base_datos>
   ```
2. Si está ejecutando una aplicación Java de UNIX en una instancia de DB2 de 64 bits, pero el SDK (software development kit) de Java es de 32 bits, es necesario que cambie la vía de acceso de la biblioteca de DB2 antes de ejecutar la aplicación. Por ejemplo, en AIX:
   - En el shell bash o Korn:

```
export LIBPATH=$HOME/sqllib/lib32
```

- En el shell C:

```
setenv LIBPATH $HOME/sqllib/lib32
```

3. Ejecute el intérprete de Java en la aplicación con este mandato:

```
java TbMod
```

Consideraciones sobre los applets Java

Puede acceder a las bases de datos DB2 utilizando applets Java.

Tenga en cuenta lo siguiente cuando utilice applets Java:

- En el caso de un applet JDBC o SQLJ de mayor tamaño que conste de varias clases Java, puede elegir empaquetar todas sus clases en un archivo JAR. Para un applet SQLJ, también tendría que empaquetar sus perfiles serializados junto con sus clases. Si decide hacer esto, añada el archivo JAR al parámetro `archive` en el código "applet". Para conocer detalles, consulte la documentación correspondiente al SDK (software development kit) de Java.

  Para los applets SQLJ, algunos navegadores no tienen todavía soporte para cargar un objeto serializado desde un archivo de recursos asociado al applet. Por ejemplo, obtendrá el mensaje de error siguiente si intenta cargar el applet de ejemplo proporcionado `Applt` en esos navegadores:

  ```
  java.lang.ClassNotFoundException: Applt_SJProfile0
  ```

  Para evitar este problema, existe un programa de utilidad que convierte un perfil serializado en un perfil que está almacenado en formato de clase Java. El programa de utilidad es una clase Java llamada sqlj.runtime.profile.util.SerProfileToClass. Este programa utiliza como entrada un archivo de recursos de un perfil serializado y produce como resultado una clase Java donde está contenido el perfil. El perfil se puede convertir utilizando uno de estos mandatos:

  ```
  profconv Applt_SJProfile0.ser
  ```

  o

  ```
  java sqlj.runtime.profile.util.SerProfileToClass Applt_SJProfile0.ser
  ```

  Como resultado se crea la clase Applt_SJProfile0.class. Normalmente el problema se resuelve sustituyendo todos los perfiles con formato `.ser` utilizados por el applet por perfiles con formato `.class`.
- Puede colocar el archivo `db2jcc.jar` en un directorio que está compartido por varios applets y que se puede cargar desde un sitio Web. `db2jcc.jar` es para applets que utilizan IBM Data Server Driver para JDBC y SQLJ o para cualquier applet SQLJ. Este archivo reside en el directorio `sqllib\java` en los sistemas operativos Windows y en el directorio `sqllib/java` en UNIX. Puede ser necesario añadir un parámetro `codebase` al código "applet" del archivo HTML para identificar el directorio. Para ver detalles, consulte la documentación del kit de desarrollo de software para Java.

  Si utiliza cualquiera de las funciones de JDBC 4.0 o posteriores, copie db2jcc4.jar en lugar de db2jcc.jar.
- El servidor de applet JDBC (receptor), `db2jd`, contiene funciones de manejo de señales para hacerlo más robusto. Como consecuencia de ello, no se puede utilizar la secuencia de teclas Control-C para concluir `db2jd`. Por tanto, la única forma de concluir el receptor es interrumpir el proceso utilizando `kill -9` (para UNIX) o el Gestor de tareas (para Windows).

Opciones de aplicaciones y applets SQLJ para UNIX

El script de creación `bldsqlj` crea aplicaciones y applets de SQLJ en los sistemas operativos UNIX. `bldsqlj` especifica un conjunto de opciones para el traductor y personalizador de SQLJ.

Recomendación: Utilice las mismas opciones para el traductor y personalizador de SQLJ que las utilizadas por `bldsqlj` cuando cree sus aplicaciones y applets SQLJ en los sistemas operativos UNIX.

Las opciones incluidas en `bldsqlj` son:

sqlj Es el traductor SQLJ (también compila el programa).

"${progname}.sqlj"
El archivo fuente de SQLJ. El mandato `progname=${1%.sqlj}` elimina la extensión si se ha incluido en el nombre de archivo de entrada, por lo que cuando se vuelve a añadir la extensión no está duplicado.

db2sqljcustomize
El personalizador de perfiles SQLJ.

-url Especifica un URL de JDBC para establecer una conexión de base de datos, como `jdbc:db2://servername:50000/sample`.

-user Especifica un ID de usuario.

-password
Especifica una contraseña.

"${progname}_SJProfile0"
Especifica un perfil serializado para el programa.

Opciones de aplicaciones y applets SQLJ para Windows

El archivo de proceso por lotes `bldsqlj.bat` crea aplicaciones y applets SQLJ en los sistemas operativos Windows. `bldsqlj.bat` especifica un conjunto de opciones para el traductor y personalizador de SQLJ.

Recomendación: Utilice las mismas opciones para el traductor y personalizador de SQLJ que las utilizadas por `bldsqlj.bat` cuando cree sus aplicaciones y applets SQLJ en los sistemas operativos Windows.

Las opciones incluidas en `bldsqlj.bat` son:

sqlj Es el traductor SQLJ (también compila el programa).

%1.sqlj
El archivo fuente de SQLJ.

db2sqljcustomize
El personalizador de perfiles SQLJ.

-url Especifica un URL de JDBC para establecer una conexión de base de datos, como `jdbc:db2://servername:50000/sample`.

-user Especifica un ID de usuario.

-password
Especifica una contraseña.

%1_SJProfile0
Especifica un perfil serializado para el programa.

Creación de rutinas SQL

Puede utilizar un archivo `makefile` de Java o el archivo de creación `bldsqljs` para crear rutinas SQLJ. Una vez creadas esas rutinas, es necesario catalogarlas.

Acerca de esta tarea

Los pasos siguientes muestran cómo crear y ejecutar el procedimiento almacenado de ejemplo `SpServer` de SQLJ. Estos pasos utilizan el archivo de creación `bldsqljs` (UNIX), o `bldsqljs.bat` (Windows), que contiene mandatos para crear un applet o aplicación SQLJ.

El archivo de creación utiliza como entrada un máximo de seis parámetros: $1, $2, $3, $4, $5 y $6 en UNIX y %1, %2, %3, %4, %5 y %6 en Windows. El primer parámetro especifica el nombre del programa. El segundo parámetro especifica el ID de usuario correspondiente a la instancia de base de datos; el tercer parámetro especifica la contraseña. El cuarto parámetro especifica el nombre del servidor. El quinto parámetro especifica el número de puerto. Finalmente, el sexto parámetro especifica el nombre de la base de datos. Se pueden utilizar valores por omisión para todos los parámetros, excepto para el primero, el nombre del programa. Consulte el archivo de creación para conocer detalles sobre la utilización de valores de parámetro por omisión.

Procedimiento

1. Cree la aplicación del procedimiento almacenado con este mandato:

   ```
   bldsqljs SpServer <ID de usuario> <contraseña> <nombre_servidor>
                                <número_puerto> <nombre_base_datos>
   ```

2. Catalogue el procedimiento almacenado con este mandato:

   ```
   spcat
   ```

   Este script conecta con la base de datos de ejemplo, descataloga mediante SpDrop.db2 las rutinas que se hubieran catalogado previamente, luego las cataloga invocando SpCreate.db2, y finalmente desconecta de la base de datos. También puede ejecutar los scripts SpDrop.db2 y SpCreate.db2 por separado.

3. Detenga y reinicie la base de datos para que se pueda reconocer el nuevo archivo de clase. Si es necesario, configure para lectura la modalidad de archivo del archivo de clase, para que pueda ser leído por el usuario delimitado.
4. Compile y ejecute la aplicación cliente `SpClient` para acceder a la clase del procedimiento almacenado. Puede crear `SpClient` mediante el archivo de creación de aplicaciones `bldsqlj` (UNIX) o `bldsqlj.bat` (Windows).

Opciones de rutinas SQLJ para UNIX

El script de creación `bldsqljs` crea rutinas de SQLJ en los sistemas operativos UNIX. `bldsqljs` especifica un conjunto de opciones para el traductor y personalizador de SQLJ.

Recomendación: Utilice las mismas opciones para el traductor y personalizador de SQLJ que las utilizadas por `bldsqljs` cuando cree sus rutinas SQLJ en las plataformas UNIX.

Las opciones incluidas en `bldsqljs` son:

`sqlj` Es el traductor SQLJ (también compila el programa).

"${progname}.sqlj"
El archivo fuente de SQLJ. El mandato `progname=${1%.sqlj}` elimina la extensión si se ha incluido en el nombre de archivo de entrada, por lo que cuando se vuelve a añadir la extensión no está duplicado.

db2sqljcustomize
El personalizador de perfiles SQLJ.

-url Especifica un URL de JDBC para establecer una conexión de base de datos, como `jdbc:db2://servername:50000/sample`.

-user Especifica un ID de usuario.

-password
Especifica una contraseña.

"${progname}_SJProfile0"
Especifica un perfil serializado para el programa.

Opciones de rutinas SQLJ para Windows

El archivo de proceso por lotes `bldsqljs.bat` crea rutinas SQLJ en los sistemas operativos Windows. `bldsqljs.bat` especifica un conjunto de opciones para el traductor y personalizador de SQLJ.

Recomendación: Utilice las mismas opciones para el traductor y personalizador de SQLJ que las utilizadas por `bldsqljs.bat` cuando cree sus rutinas SQLJ en los sistemas operativos Windows.

Las opciones siguientes del personalizador y el conversor SQLJ se utilizan en el archivo por lotes `bldsqljs.bat` en los sistemas operativos Windows. Estas son las opciones que DB2 recomienda utilizar para crear rutinas SQLJ (procedimientos almacenados y funciones definidas por el usuario).

sqlj Es el traductor SQLJ (también compila el programa).

%1.sqlj
El archivo fuente de SQLJ.

db2sqljcustomize
El personalizador de perfiles de DB2 para Java.

-url Especifica un URL de JDBC para establecer una conexión de base de datos, como `jdbc:db2://servername:50000/sample`.

-user Especifica un ID de usuario.

-password
Especifica una contraseña.

%1_SJProfile0
Especifica un perfil serializado para el programa.

Capítulo 7. Diagnóstico de problemas con IBM Data Server Driver para JDBC y SQLJ

IBM Data Server Driver para JDBC y SQLJ incluye las herramientas de diagnóstico y los rastreos necesarios para diagnosticar problemas durante la conexión y la ejecución de sentencias de SQL.

Prueba de la conexión de servidor de datos

Ejecute el programa de utilidad DB2Jcc para probar una conexión con un servidor de datos. Especifique DB2Jcc con el URL para el servidor de datos, para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 o IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2. DB2Jcc intenta conectar con el servidor de datos y ejecutar una sentencia de SQL y un método DatabaseMetaData. Si falla la conexión o la ejecución de sentencias, DB2Jcc proporciona información de diagnóstico sobre el error.

Recogida de datos de rastreo de JDBC

Utilice uno de los procedimientos siguientes para iniciar el rastreo:

Procedimiento 1: Para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 o IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 para DB2 para Linux, UNIX y Windows, el método recomendado es iniciar el rastreo estableciendo la propiedad db2.jcc.override.traceFile o la propiedad db2.jcc.override.traceDirectory en el archivo de propiedades de configuración de IBM Data Server Driver para JDBC y SQLJ. Puede establecer las propiedades db2.jcc.tracePolling y db2.jcc.tracePollingInterval antes de iniciar el controlador que le permita cambiar las propiedades de rastreo de configuración global mientras el controlador esté en ejecución.

Procedimiento 2: Si utiliza la interfaz DataSource para conectar con una fuente de datos, realice lo siguiente para iniciar el rastreo:

1. Invoque el método DB2BaseDataSource.setTraceLevel para determinar el tipo de rastreo que necesita. El nivel de rastreo por omisión es `TRACE_ALL`. Consulte "Propiedades de IBM Data Server Driver para JDBC y SQLJ" para obtener información sobre cómo especificar más de un tipo de rastreo.
2. Invoque el método DB2BaseDataSource.setJccLogWriter para especificar el destino del rastreo y activar el rastreo.

Procedimiento 3:

Si utiliza la interfaz DataSource para conectar con una fuente de datos, invoque el método javax.sql.DataSource.setLogWriter para activar el rastreo. Con este método, `TRACE_ALL` es el único nivel de rastreo disponible.

Si utiliza la interfaz DriverManager para conectar con una fuente de datos, lleve a cabo el procedimiento siguiente para iniciar el rastreo:

1. Invoque el método DriverManager.getConnection con la propiedad `traceLevel` establecida en el parámetro *info* o en el parámetro *url* para el tipo de rastreo que necesita. El nivel de rastreo por omisión es `TRACE_ALL`. Consulte

"Propiedades de IBM Data Server Driver para JDBC y SQLJ" para obtener información sobre cómo especificar más de un tipo de rastreo.

2. Invoque el método DriverManager.setLogWriter para especificar el destino del rastreo y activar el rastreo.

Después de que se establezca una conexión, puede desactivar el rastreo o volverlo a activar, cambiar el destino del rastreo o cambiar el nivel de rastreo mediante el método DB2Connection.setJccLogWriter. Para desactivar el rastreo, establezca el valor de logWriter en `null`.

La propiedad logWriter es un objeto de tipo java.io.PrintWriter. Si la aplicación no puede manejar objetos java.io.PrintWriter, puede utilizar la propiedad `traceFile` para especificar el destino de la salida del rastreo. Para utilizar la propiedad `traceFile`, establezca la propiedad logWriter en `null` y establezca la propiedad `traceFile` con el nombre del archivo en que el controlador graba los datos del rastreo. Se tiene que poder grabar en este archivo y en el directorio en que reside. Si el archivo ya existe, el controlador lo sobregraba.

Procedimiento 4: Si está utilizando la interfaz DriverManager, especifique las propiedades `traceFile` y `traceLevel` como parte del URL cuando cargue el controlador. Por ejemplo:

```
String url = "jdbc:db2://sysmvs1.stl.ibm.com:5021/san_jose" +
 ":traceFile=/u/db2p/jcctrace;" +
 "traceLevel=" + com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DRDA_FLOWS + ";";
```

Procedimiento 5: Utilice métodos DB2TraceManager. La clase DB2TraceManager proporciona la capacidad para suspender y reanudar el rastreo de cualquier tipo de programa de anotaciones cronológicas.

Ejemplo de inicio de un rastreo utilizando propiedades de configuración: Para ver un ejemplo completo de cómo utilizar parámetros de configuración para recoger datos de rastreo, consulte "Ejemplo de utilización de propiedades de configuración para iniciar un rastreo JDBC".

Programa de rastreo de ejemplo: Para ver un ejemplo completo de un programa de rastreo que se ejecuta bajo IBM Data Server Driver para JDBC y SQLJ, vea "Ejemplo de un programa de rastreo que se ejecuta bajo IBM Data Server Driver para JDBC y SQLJ".

Recogida de datos de rastreo de SQLJ durante la personalización o vinculación

Para recopilar datos de rastreo con el fin de diagnosticar problemas durante el proceso de personalización o vinculación de SQLJ, especifique las opciones -tracelevel y -tracefile cuando ejecute db2sqljcustomize o el programa de utilidad de vinculación db2sqljbind.

Formato de información sobre un perfil serializado de SQLJ

El programa de utilidad `profp` formatea la información sobre cada una de las cláusulas SQLJ de un perfil serializado. El formato del programa de utilidad `profp` es:

Ejecute el programa de utilidad `profp` sobre el perfil serializado correspondiente a la conexión donde se produzca el error. Si se emite una excepción, se genera un rastreo de Java. A partir del rastreo de pila, puede determinar qué perfil serializado se estaba utilizando cuando se emitió la excepción.

Formato de información sobre un perfil serializado personalizado de SQLJ

El programa de utilidad `db2sqljprint` da formato a la información sobre cada cláusula de SQLJ contenida en un perfil serializado que esté personalizado para IBM Data Server Driver para JDBC y SQLJ.

Ejecute el programa de utilidad `db2sqljprint` sobre el perfil serializado personalizado correspondiente a la conexión donde se produzca el error.

DB2Jcc - Programa de utilidad de diagnóstico de IBM Data Server Driver para JDBC y SQLJ

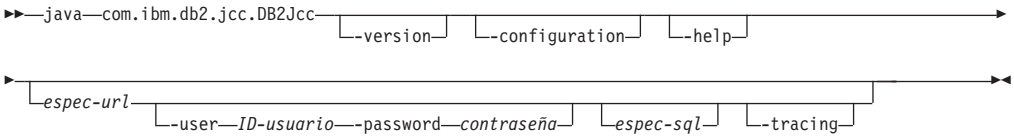
DB2Jcc comprueba si un servidor de datos está configurado para el acceso a base de datos.

Para comprobar la conexión, DB2Jcc conecta con el servidor de datos especificado, ejecuta una sentencia de SQL y ejecuta un método java.sql.DatabaseMetadata.

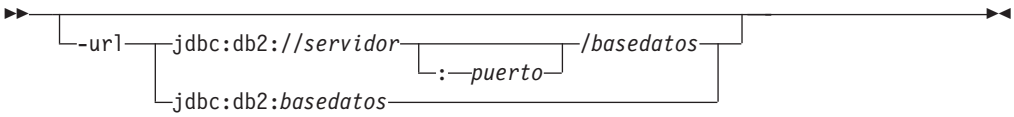
Autorización

El ID de usuario en el que se ejecuta DB2Jcc debe tener la autorización necesaria para conectar con el servidor de datos especificado y ejecutar la sentencia de SQL indicada.

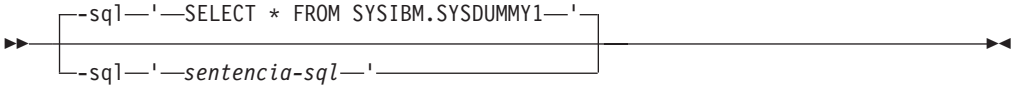
Sintaxis de DB2Jcc



espec-url:



espec-sql:



Parámetros de DB2Jcc

-help
Especifica que DB2Jcc describa todas las opciones a las que da soporte. Si se especifica cualquier otra opción con -help, no se tiene en cuenta.

-version
Especifica que DB2Jcc muestre el nombre y la versión del controlador.

-configuration
Especifica que DB2Jcc muestre la información de configuración del controlador.

-url
Especifica el URL para el servidor de datos cuya conexión se está probando. El URL puede ser un URL para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 o IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4. Las partes variables del valor `-url` son:

servidor
Nombre de dominio o dirección IP del sistema operativo donde reside el servidor de bases de datos. *servidor* se utiliza únicamente para la conectividad de tipo 4.

puerto
El número de puerto del servidor TCP/IP que está asignado al servidor de datos. El valor por omisión es 446. *puerto* se utiliza únicamente para la conectividad de tipo 4.

basedatos
Nombre del servidor de bases de datos para el que se va a personalizar el perfil.

Si la conexión es con un servidor DB2 para z/OS, *basedatos* es el nombre de ubicación de DB2 que se define durante la instalación. Todos los caracteres de este valor deben ser caracteres en mayúsculas. Puede determinar el nombre de ubicación ejecutando la sentencia de SQL siguiente en el servidor:

```
SELECT CURRENT SERVER FROM SYSIBM.SYSDUMMY1;
```

Si la conexión es con un servidor DB2 Database para Linux, UNIX y Windows, *basedatos* es el nombre de la base de datos que se define durante la instalación.

Si la conexión es con un servidor de datos IBM Informix, *basedatos* es el nombre de la base de datos. El nombre no es sensible a las mayúsculas y las minúsculas. El servidor convierte el nombre a minúsculas.

Si la conexión es con un servidor IBM Cloudscape, *basedatos* es el nombre totalmente calificado del archivo donde reside la base de datos. Este nombre se debe incluir entre comillas dobles ("). Por ejemplo:

```
"c:/basedatos/testdb"
```

-user *ID-usuario*
Especifica el ID de usuario que se utilizará para probar la conexión al servidor de datos.

-password *contraseña*
Especifica la contraseña del ID de usuario que se utilizará para probar la conexión al servidor de datos.

-sql '*sentencia-sql***'**
Especifica la sentencia de SQL que se envía al servidor de datos para comprobar la conexión. Si no se indica el parámetro -sql esta sentencia de SQL se envía al servidor de datos:

```
SELECT * FROM SYSIBM.SYSDUMMY1
```

-tracing
Especifique que el rastreo está habilitado. El destino del rastreo es System.out.

Si se omite el parámetro -tracing, el rastreo se inhabilita.

Ejemplos

Ejemplo: Pruebe la conexión con el servidor de datos mediante IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4. Utilice la sentencia de SQL por omisión para probar la conexión. Habilite el rastreo para la prueba.

```
java com.ibm.db2.jcc.DB2Jcc
-url jdbc:db2://mysys.myloc.svl.ibm.com:446/MYDB
-user db2user -password db2pass -tracing
```

Ejemplo: Pruebe la conexión con el servidor de datos mediante IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2. Utilice la sentencia de SQL siguiente para probar la conexión:

```
SELECT COUNT(*) FROM EMPLOYEE
```

Inhabilite el rastreo para la prueba.

```
java com.ibm.db2.jcc.DB2Jcc
-url jdbc:db2:MYDB
-user db2user -password db2pass
-sql 'SELECT COUNT(*) FROM EMPLOYEE'
```

Ejemplos de utilización de propiedades de configuración para iniciar un rastreo de JDBC

Puede controlar el rastreo de aplicaciones JDBC sin modificar esas aplicaciones.

Ejemplo de grabación de datos de rastreo en un archivo de rastreo para cada conexión

Suponga que desea reunir datos de rastreo para un programa llamado Test.java, el cual utiliza IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4. Test.java no realiza ningún rastreo y no es recomendable modificar el programa. Por lo tanto, el rastreo se habilita mediante las propiedades de configuración. Suponga que desea que la salida del rastreo tenga las características siguientes:

- La información de rastreo de cada conexión del mismo DataSource se graba en un archivo de rastreo diferente. La salida se coloca en un directorio llamado /Trace.
- El nombre de cada archivo de rastreo empieza por jccTrace1.
- Si los archivos de rastreo que tienen los mismos nombres ya existen, los datos del rastreo se añadirán a éstos.

Aunque Test.java no contenga ningún código para llevar a cabo el rastreo, es recomendable definir las propiedades de configuración de modo que si la aplicación se modifica en el futuro para llevar a cabo el rastreo, los valores del

programa prevalecerán sobre los valores de las propiedades de configuración. Para ello, utilice el conjunto de propiedades de configuración que empiezan por db2.jcc, no por db2.jcc.override.

Los valores de las propiedades de configuración tienen el aspecto siguiente:
- db2.jcc.traceDirectory=/Trace
- db2.jcc.traceFile=jccTrace1
- db2.jcc.traceFileAppend=true

Es recomendable que los valores del rastreo se apliquen solamente al programa autónomo denominado Test.java. Por lo tanto, cree un archivo con dichos valores y, a continuación, haga referencia al archivo al invocar el programa de Java especificando la opción -Ddb2.jcc.propertiesFile. Suponga que el archivo que contiene los valores es /Test/jcc.properties. Para habilitar el rastreo al ejecutar Test.java, deberá emitir un mandato como el siguiente:

```
java -Ddb2.jcc.propertiesFile=/Test/jcc.properties Test
```

Suponga que Test.java crea dos conexiones para un DataSource. El programa no define ningún objeto logWriter; por lo tanto, el controlador crea un objeto logWriter global para la salida del rastreo. Cuando el programa finaliza, los archivos siguientes contienen los datos del rastreo:
- /Trace/jccTrace1_global_0
- /Trace/jccTrace1_global_1

Ejemplo de realización de rastreo circular con un número fijo de archivos y un tamaño de archivo fijo

Suponga que desea reunir datos de rastreo para un programa llamado Test.java, el cual utiliza IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4. Test.java no realiza ningún rastreo y no es recomendable modificar el programa. Por lo tanto, el rastreo se habilita mediante las propiedades de configuración. Suponga que desea que la salida del rastreo tenga las características siguientes:
- La información de rastreo de cada conexión del mismo DataSource se graba en un conjunto de archivos de rastreo independiente.
- El número máximo de archivos de rastreo que se graban para cada conexión es 4.
- Cuando todos los archivos de rastreo están llenos, el rastreo sobrescribe los datos de rastreo existentes, empezando por el primer archivo de rastreo que se haya grabado.
- El tamaño máximo de cada archivo de rastreo es 4 MB.
- El nombre de los archivos de rastreo empieza por jcc.log y se graba en un directorio denominado /Trace.
- Si ya existen archivos de rastreo con los mismos nombres, los datos de rastreo se sobrescriben.

Aunque Test.java no contenga ningún código para llevar a cabo el rastreo, es recomendable definir las propiedades de configuración de modo que si la aplicación se modifica en el futuro para llevar a cabo el rastreo, los valores del programa prevalecerán sobre los valores de las propiedades de configuración. Para ello, utilice el conjunto de propiedades de configuración que empiezan por db2.jcc.

Los valores de las propiedades de configuración tienen el aspecto siguiente:
- db2.jcc.traceFile=jcc.log
- db2.jcc.traceOption=1

- db2.jcc.traceFileSize=4194304
- db2.jcc.traceFileCount=4
- db2.jcc.traceFileAppend=false

Es recomendable que los valores del rastreo se apliquen solamente al programa autónomo denominado Test.java. Por lo tanto, cree un archivo con dichos valores y, a continuación, haga referencia al archivo al invocar el programa de Java especificando la opción -Ddb2.jcc.propertiesFile. Suponga que el archivo que contiene los valores es /Test/jcc.properties. Para habilitar el rastreo al ejecutar Test.java, deberá emitir un mandato como el siguiente:

```
java -Ddb2.jcc.propertiesFile=/Test/jcc.properties Test
```

Suponga que Test.java crea dos conexiones para un DataSource. El programa no define ningún objeto logWriter; por lo tanto, el controlador crea un objeto logWriter global para la salida del rastreo. Durante la ejecución del programa, IBM Data Server Driver para JDBC y SQLJ graba 17 MB de datos para la primera conexión y 10 MB de datos para la segunda conexión.

Cuando el programa finaliza, los archivos siguientes contienen los datos del rastreo:
- /Trace/jcc.log_global_0.1
- /Trace/jcc.log_global_0.2
- /Trace/jcc.log_global_0.3
- /Trace/jcc.log_global_0.4
- /Trace/jcc.log_global_1.1
- /Trace/jcc.log_global_1.2
- /Trace/jcc.log_global_1.3

/Trace/jcc.log_global_0.1 contiene el último MB de datos de rastreo que se graba para la primera conexión, que sobrescribe el primer MB datos de rastreo que se ha grabado para dicha conexión.

Ejemplo de un programa de rastreo que se ejecuta bajo IBM Data Server Driver para JDBC y SQLJ

Puede ser conveniente escribir una clase individual que incluya métodos para realizar rastreos bajo la interfaz DriverManager así como la interfaz DataSource.

El ejemplo siguiente muestra una clase con esas características. El ejemplo utiliza IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4.

Figura 46. Ejemplo de rastreo ejecutado bajo IBM Data Server Driver para JDBC y SQLJ

```
public class TraceExample
{

  public static void main(String[] args)
  {
    sampleConnectUsingSimpleDataSource();
    sampleConnectWithURLUsingDriverManager();
  }

  private static void sampleConnectUsingSimpleDataSource()
  {
    java.sql.Connection c = null;
    java.io.PrintWriter printWriter =
     new java.io.PrintWriter(System.out, true);
```

```
                                        // Imprime en consola, true significa
                                        // desecho automático para
                                        // no perder rastreo
  try {
    javax.sql.DataSource ds =
     new com.ibm.db2.jcc.DB2SimpleDataSource();
    ((com.ibm.db2.jcc.DB2BaseDataSource) ds).setServerName("sysmvs1.stl.ibm.com");
    ((com.ibm.db2.jcc.DB2BaseDataSource) ds).setPortNumber(5021);
    ((com.ibm.db2.jcc.DB2BaseDataSource) ds).setDatabaseName("san_jose");
    ((com.ibm.db2.jcc.DB2BaseDataSource) ds).setDriverType(4);

   ds.setLogWriter(printWriter);    // Esto activa el rastreo

   // Refinar el nivel de detalle del rastreo
   ((com.ibm.db2.jcc.DB2BaseDataSource) ds).
    setTraceLevel(com.ibm.db2.jcc.DB2SimpleDataSource.TRACE_CONNECTS |
    com.ibm.db2.jcc.DB2SimpleDataSource.TRACE_DRDA_FLOWS);

   // Esta petición de conexión se rastreará utilizando el nivel de rastreo
   // TRACE_CONNECTS | TRACE_DRDA_FLOWS
   c = ds.getConnection("myname", "mypass");

   // Cambiar el nivel de rastreo a TRACE_ALL
   // para todas las peticiones subsiguientes de la conexión
   ((com.ibm.db2.jcc.DB2Connection) c).setJccLogWriter(printWriter,
    com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL);
   // La sentencia INSERT siguiente se rastrea
   // utilizando el nivel de rastreo TRACE_ALL
   java.sql.Statement s1 = c.createStatement();
   s1.executeUpdate("INSERT INTO sampleTable(sampleColumn) VALUES(1)");
   s1.close();

   // El código siguiente inhabilita todo el rastreo de la conexión
   ((com.ibm.db2.jcc.DB2Connection) c).setJccLogWriter(null);

   // La sentencia INSERT siguiente no se rastrea
   java.sql.Statement s2 = c.createStatement();
   s2.executeUpdate("INSERT INTO sampleTable(sampleColumn) VALUES(1)");
   s2.close();

   c.close();
  }
  catch(java.sql.SQLException e) {
   com.ibm.db2.jcc.DB2ExceptionFormatter.printTrace(e,
    printWriter, "[TraceExample]");
  }
  finally {
    cleanup(c, printWriter);
    printWriter.flush();
  }
}

// Si el código se ha ejecutado satisfactoriamente, la conexión
// ya debe estar cerrada. Comprobar si lo está.
// Si la conexión está cerrada, simplemente finalice el programa.
// Si se ha producido un error, intente retrotraer (rollback)
// y cierre la conexión.

private static void cleanup(java.sql.Connection c,
 java.io.PrintWriter printWriter)
{
  if(c == null) return;

  try {
   if(c.isClosed()) {
      printWriter.println("[TraceExample] " +
       "La conexión se ha cerrado satisfactoriamente");
```

```
      return;
    }

    // Si se ha llegado aquí, algo ha ido mal.
    // Retrotraer y cerrar la conexión.
    printWriter.println("[TraceExample] Se está retrotrayendo la conexión");
    try {
      c.rollback();
    }
    catch(java.sql.SQLException e) {
      printWriter.println("[TraceExample] " +
       "Se capturó la siguiente java.sql.SQLException al intentar retrotraer:");
      com.ibm.db2.jcc.DB2ExceptionFormatter.printTrace(e, printWriter,
       "[TraceExample]");
      printWriter.println("[TraceExample] " +
       "No se puede retrotraer la conexión");
     }
     catch(java.lang.Throwable e) {
       printWriter.println("[TraceExample] Se capturó " +
        "la siguiente java.lang.Throwable al intentar retrotraer:");
       com.ibm.db2.jcc.DB2ExceptionFormatter.printTrace(e,
        printWriter, "[TraceExample]");
       printWriter.println("[TraceExample] No se puede " +
        "roll back the connection");
     }

     // Cierre la conexión
     printWriter.println("[TraceExample] Se está cerrando la conexión");
     try {
       c.close();
     }
     catch(java.sql.SQLException e) {
       printWriter.println("[TraceExample] Excepción al " +
        "intentar cerrar la conexión");
       printWriter.println("[TraceExample] Pueden producirse " +
        "puntos muertos si no se cierra la conexión.");
       com.ibm.db2.jcc.DB2ExceptionFormatter.printTrace(e, printWriter,
        "[TraceExample]");
     }
     catch(java.lang.Throwable e) {
       printWriter.println("[TraceExample] Se ha emitido Throwable " +
        "al intentar cerrar la conexión");
       printWriter.println("[TraceExample] Pueden producirse " +
        "puntos muertos si no se cierra la conexión.");
       com.ibm.db2.jcc.DB2ExceptionFormatter.printTrace(e, printWriter,
        "[TraceExample]");
     }
   }
   catch(java.lang.Throwable e) {
     printWriter.println("[TraceExample] No se puede " +
      "forzar el cierre de la conexión");
     printWriter.println("[TraceExample] Pueden producirse " +
      "puntos muertos si no se cierra la conexión.");
     com.ibm.db2.jcc.DB2ExceptionFormatter.printTrace(e, printWriter,
      "[TraceExample]");
   }
}
private static void sampleConnectWithURLUsingDriverManager()
{
  java.sql.Connection c = null;

  // Esta vez, enviar printWriter a un archivo.
  java.io.PrintWriter printWriter = null;
  try {
    printWriter =
      new java.io.PrintWriter(
        new java.io.BufferedOutputStream(
```

```
        new java.io.FileOutputStream("/temp/driverLog.txt"), 4096), true);
}
catch(java.io.FileNotFoundException e) {
  java.lang.System.err.println
  ("No se puede definir un transcriptor de impresión para el rastreo");
  java.lang.System.err.flush();
  return;
}

try {
  Class.forName("com.ibm.db2.jcc.DB2Driver");
}
catch(ClassNotFoundException e) {
  printWriter.println("[TraceExample] " +
   "Conectividad de IBM Data Server Driver para JDBC y SQLJ tipo 4 " +
   "no está en classpath de la aplicación. No se puede cargar el controlador.");
  printWriter.flush();
  return;
}

// Este URL describe fuente de datos de destino para conectividad de tipo 4.
// La propiedad traceLevel se establece mediante la sintaxis de URL y
// el rastreo del controlador se dirige al archivo "/temp/driverLog.txt"
// La propiedad traceLevel tiene un tipo int. Las constantes
// com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DRDA_FLOWS y
// com.ibm.db2.jcc.DB2BaseDataSource.TRACE_CONNECTS representan
// valores int. Estas constantes no se pueden usar directamente en el
// primer parámetro getConnection. Resuelva las constantes a sus
// valores int asignándolos a una variable. A continuación utilice la
// variable como el primer parámetro del método getConnection.
String databaseURL =
 "jdbc:db2://sysmvs1.stl.ibm.com:5021" +
 "/sample:traceFile=/temp/driverLog.txt;traceLevel=" +
 (com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DRDA_FLOWS |
 com.ibm.db2.jcc.DB2BaseDataSource.TRACE_CONNECTS) + ";";

// Definir otras propiedades
java.util.Properties properties = new java.util.Properties();
properties.setProperty("user", "myname");
properties.setProperty("password", "mypass");

try {
 // Esta petición de conexión se rastreará utilizando el nivel de rastreo
 // TRACE_CONNECTS | TRACE_DRDA_FLOWS
 c = java.sql.DriverManager.getConnection(databaseURL, properties);

 // Cambiar el nivel de rastreo para todas las peticiones posteriores
 // de la conexión por TRACE_ALL
 ((com.ibm.db2.jcc.DB2Connection) c).setJccLogWriter(printWriter,
  com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL);

 // La sentencia INSERT siguiente se rastrea
 // utilizando el nivel de rastreo TRACE_ALL
 java.sql.Statement s1 = c.createStatement();
 s1.executeUpdate("INSERT INTO sampleTable(sampleColumn) VALUES(1)");
 s1.close();

 // Inhabilite todo el rastreo de la conexión
 ((com.ibm.db2.jcc.DB2Connection) c).setJccLogWriter(null);

  // El siguiente código de inserción de SQL no se rastrea
  java.sql.Statement s2 = c.createStatement();
  s2.executeUpdate("insert into sampleTable(sampleColumn) values(1)");
  s2.close();

  c.close();
}
```

```
    catch(java.sql.SQLException e) {
     com.ibm.db2.jcc.DB2ExceptionFormatter.printTrace(e, printWriter,
      "[TraceExample]");
    }
    finally {
      cleanup(c, printWriter);
      printWriter.flush();
    }
  }
}
```

Técnicas para supervisar el soporte de Sysplex de IBM Data Server Driver para JDBC y SQLJ

Para supervisar el soporte de Sysplex de IBM Data Server Driver para JDBC y SQLJ, hay que supervisar la agrupación de objetos de transporte global.

La agrupación de objetos de transporte global se puede supervisar de cualquiera de los modos siguientes:

- Mediante rastreos que se inician estableciendo las propiedades de configuración de IBM Data Server Driver para JDBC y SQLJ
- Mediante una interfaz de programación de aplicaciones

Configuración de propiedades para supervisar la agrupación de objetos de transporte global

Las propiedades de configuración db2.jcc.dumpPool, db2.jcc.dumpPoolStatisticsOnSchedule, y db2.jcc.dumpPoolStatisticsOnScheduleFile controlan el rastreo de la agrupación de objetos de transporte global.

Por ejemplo, el siguiente conjunto de valores de propiedades de configuración produce mensajes de error y mensajes de error de agrupación de vuelco para ser escritos cada 60 segundos en un archivo llamado /home/WAS/logs/srv1/poolstats:

```
db2.jcc.dumpPool=DUMP_SYSPLEX_MSG|DUMP_POOL_ERROR
db2.jcc.dumpPoolStatisticsOnSchedule=60
db2.jcc.dumpPoolStatisticsOnScheduleFile=/home/WAS/logs/srv1/poolstats
```

Una entrada del archivo de estadísticas de agrupación tiene el aspecto siguiente:

```
time Scheduled PoolStatistics npr:2575 nsr:2575 lwroc:439 hwroc:1764 coc:372
aooc:362 rmoc:362 nbr:2872 tbt:857520 tpo:10
```

Los significados de los campos son:

npr
: Número total de peticiones que IBM Data Server Driver para JDBC y SQLJ ha realizado a la agrupación desde la creación de ésta.

nsr
: Número de peticiones satisfactorias que el IBM Data Server Driver para JDBC y SQLJ ha realizado a la agrupación desde la creación de ésta. Una petición satisfactoria indica que la agrupación ha devuelto un objeto.

lwroc
: Número de objetos que se han vuelto a utilizar pero que no se encontraban en la agrupación. Esto puede ocurrir si un objeto Connection libera un objeto de transporte en el límite de una transacción. Si el objeto Connection necesita un

objeto de transporte posteriormente y ningún otro objeto Connection ha utiliza el objeto de transporte original, el objeto Connection podrá utilizar el objeto de transporte.

hwroc
Número de objetos de la agrupación que se han vuelto a utilizar.

coc
Número de objetos que IBM Data Server Driver para JDBC y SQLJ ha creado desde la creación de la agrupación.

aooc
Número de objetos que han excedido el tiempo de inactividad especificado por db2.jcc.maxTransportObjectIdleTime y que no se han suprimido de la agrupación.

rmoc
Número de objetos que se han suprimido de la agrupación desde que ésta fue creada.

nbr
Número de peticiones que IBM Data Server Driver para JDBC y SQLJ ha realizado a la agrupación que la agrupación bloqueó debido a que ésta había llegado a su capacidad máxima. Es posible que una petición bloqueada se ejecute correctamente si el objeto se devuelve a la agrupación antes de que se supere el tiempo especificado por db2.jcc.maxTransportObjectWaitTime y que se haya emitido una excepción.

tbt
Tiempo total en milisegundos que la agrupación ha empleado para bloquear las peticiones. Este tiempo puede ser mucho mayor que el tiempo de ejecución de la aplicación transcurrido si la aplicación utiliza varias hebras.

sbt
Tiempo más breve en milisegundos que una hebra ha esperado hasta obtener un objeto de la agrupación. Si el tiempo es inferior a un milisegundo, el valor de este campo será cero.

lbt
Tiempo más largo en milisegundos que una hebra ha esperado hasta obtener un objeto de la agrupación.

abt
Promedio de tiempo en milisegundos que las hebras han esperado hasta obtener un objeto de transporte de la agrupación. Este valor es tbt/nbr.

tpo
Número de objetos que se encuentran actualmente en la agrupación.

Interfaces de programación de aplicaciones para la supervisión de la agrupación de objetos de transporte global

Es posible grabar aplicaciones para recopilar estadísticas sobre la agrupación de objetos de transporte global. Dichas aplicaciones crean objetos en la clase DB2PoolMonitor e invocan métodos para recuperar información acerca de la agrupación.

Por ejemplo, el código siguiente crea un objeto para supervisar la agrupación de objetos de transporte global:

```
import com.ibm.db2.jcc.DB2PoolMonitor;
DB2PoolMonitor transportObjectPoolMonitor =
        DB2PoolMonitor.getPoolMonitor (DB2PoolMonitor.TRANSPORT_OBJECT);
```

Una vez que haya creado el objeto DB2PoolMonitor, podrá utilizar los métodos siguientes en la clase DB2PoolMonitor para supervisar la agrupación.

Capítulo 8. Supervisión del sistema para IBM Data Server Driver para JDBC y SQLJ

Para ayudarle a supervisar el rendimiento de sus aplicaciones mediante IBM Data Server Driver para JDBC y SQLJ, el controlador proporciona dos métodos para recoger información sobre una conexión.

Esa información es:

Tiempo del controlador básico
: La suma de los tiempos transcurridos de API supervisada que se recogieron mientras la supervisión del sistema estaba habilitada, en microsegundos. En general, solamente se supervisan las API que podrían dar lugar a una interacción de E/S de red o del servidor de bases de datos.

Tiempo de E/S de red
: La suma de los tiempos transcurridos de E/S de red que se recogieron mientras la supervisión del sistema estaba habilitada, en microsegundos.

Tiempo del servidor
: Suma de todos los tiempos transcurridos notificados del servidor de bases de datos que se han recogido mientras estaba habilitada la supervisión del sistema, expresado en microsegundos.

Tiempo de la aplicación
: Suma de los tiempos transcurridos de la aplicación, controlador JDBC, E/S de red y servidor de bases de datos, expresado en milisegundos.

Los dos métodos son:

- La interfaz DB2SystemMonitor
- El nivel de rastreo TRACE_SYSTEM_MONITOR

Para recoger datos de supervisión del sistema utilizando la interfaz DB2SystemMonitor siga estos pasos básicos:

1. Invoque el método DB2Connection.getDB2SystemMonitor para crear un objeto DB2SystemMonitor.
2. Invoque el método DB2SystemMonitor.enable para habilitar el objeto DB2SystemMonitor para la conexión.
3. Invoque el método DB2SystemMonitor.start para iniciar la supervisión del sistema.
4. Cuando la actividad que supervisar se complete, invoque DB2SystemMonitor.stop para detener la supervisión del sistema.
5. Invoque los métodos DB2SystemMonitor.getCoreDriverTimeMicros, DB2SystemMonitor.getNetworkIOTimeMicros, `DB2SystemMonitor.getServerTimeMicros` o DB2SystemMonitor.getApplicationTimeMillis para recuperar los datos del tiempo transcurrido.

   El tiempo de servidor que devuelve `DB2SystemMonitor.getServerTimeMicros` incluye el tiempo de confirmación y retrotracción.

Por ejemplo, el código siguiente demuestra cómo recoger cada tipo de dato de tiempo transcurrido. Los números que aparecen a la derecha de algunas sentencias

corresponden a los pasos descritos anteriormente.

```
import java.sql.*;
import com.ibm.db2.jcc.*;
public class TestSystemMonitor
{
  public static void main(String[] args)
  {
    String url = "jdbc:db2://sysmvs1.svl.ibm.com:5021/san_jose";
    String user = "db2adm";
    String password = "db2adm";
    try
    {
      // Cargar IBM Data Server Driver para JDBC y SQLJ
      Class.forName("com.ibm.db2.jcc.DB2Driver");
      System.out.println("**** Controlador JDBC cargado");

      // Crear conexión utilizando IBM Data Server Driver para JDBC y SQLJ
      Connection conn = DriverManager.getConnection (url,user,password);
      // Confirmar los cambios manualmente
      conn.setAutoCommit(false);
      System.out.println("**** Creada una conexión JDBC con la fuente de datos");
      DB2SystemMonitor systemMonitor =                                    1
        ((DB2Connection)conn).getDB2SystemMonitor();
      systemMonitor.enable(true);                                         2
      systemMonitor.start(DB2SystemMonitor.RESET_TIMES);                  3
      Statement stmt = conn.createStatement();
      int numUpd = stmt.executeUpdate(
        "UPDATE EMPLOYEE SET PHONENO='4657' WHERE EMPNO='000010'");
      systemMonitor.stop();                                               4
      System.out.println("Tiempo transcurrido de servidor (microsegundos)="
        + systemMonitor.getServerTimeMicros());                           5
      System.out.println("Tiempo transcurrido de E/S de red (microsegundos)="
        + systemMonitor.getNetworkIOTimeMicros());
      System.out.println("Tiempo transcurrido controlador básico (microsegundos)="
        + systemMonitor.getCoreDriverTimeMicros());
      System.out.println("Tiempo transcurrido de aplicación (milisegundos)="
        + systemMonitor.getApplicationTimeMillis());
      conn.rollback();
      stmt.close();
      conn.close();
    }
    // Manejar los errores
    catch (ClassNotFoundException e)
    {
      System.err.println("No se puede cargar el controlador, " + e);
    }
    catch (SQLException e)
    {
      System.out.println("SQLException: " + e);
      e.printStackTrace();
    }
  }
}
```

Figura 47. Ejemplo de utilización de métodos DB2SystemMonitor para recoger datos de supervisión del sistema

Para recoger información de supervisión del sistema utilizando el método de rastreo: Inicie un rastreo JDBC, utilizando propiedades de configuración o las propiedades Connection o DataSource. Incluya TRACE_SYSTEM_MONITOR al establecer la propiedad traceLevel. Por ejemplo:

```
String url = "jdbc:db2://sysmvs1.stl.ibm.com:5021/san_jose" +
 ":traceFile=/u/db2p/jcctrace;" +
 "traceLevel=" + com.ibm.db2.jcc.DB2BaseDataSource.TRACE_SYSTEM_MONITOR + ";";
```

Los registros de rastreo con información de supervisión del sistema son parecidos a este:

```
[jcc][SystemMonitor:start]
...
[jcc][SystemMonitor:stop] core: 565.67ms | network: 211.695ms | server: 207.771ms
```

Controlador de rastreo remoto de IBM Data Server Driver para JDBC y SQLJ

IBM Data Server Driver para JDBC y SQLJ proporciona un recurso para controlar dinámicamente los rastreos de IBM Data Server Driver para JDBC y SQLJ.

Este controlador de rastreo remoto le permite efectuar operaciones tales como las siguientes para varias instancias del controlador:

- Iniciar, detener o reanudar un rastreo
- Cambiar la ubicación del archivo o directorio de rastreo de salida
- Cambiar el nivel de rastreo

El controlador de rastreo remoto utiliza la arquitectura Java Management Extensions (JMX), que forma parte de Java Standard Edition, Versión 6 o posterior. JMX consta de lo siguiente:

- Un conjunto de programas de utilidad de gestión incorporados, que le permiten realizar operaciones de supervisión desde una consola de gestión, como la consola de supervisión y gestión de Java (JConsole).
- Un conjunto de interfaces API que le permiten escribir aplicaciones para ejecutar las mismas funciones.

Habilitación del controlador de rastreo remoto

Habilitar el controlador de rastreo remoto supone habilitar Java Management Extensions (JMX) en IBM Data Server Driver para JDBC y SQLJ, y hacer que el agente JMX esté disponible para los clientes.

Antes de empezar

El controlador de rastreo remoto necesita Java Standard Edition, Versión 6 o posterior.

Acerca de esta tarea

Siga estos pasos para habilitar el controlador de rastreo remoto:

Procedimiento

1. Habilite JMX para IBM Data Server Driver para JDBC y SQLJ estableciendo la propiedad de configuración global db2.jcc.jmxEnabled en true o yes.

   Por ejemplo, incluya esta serie de caracteres en DB2JccConfiguration.properties:

   ```
   db2.jcc.jmxEnabled=true
   ```

2. Haga que el agente JMX (servidor MBean de la plataforma) esté disponible para los clientes locales o remotos.
   - Para los clientes locales:

las funciones de supervisión y gestión pasan automáticamente a estar disponibles cuando se inicia la JVM. Después de iniciar su aplicación, puede utilizar un cliente JMX tal como JConsole para conectar localmente con su proceso Java.

- Para los clientes remotos, utilice uno de estos métodos:
  - Utilice el agente JMX tal como se proporciona.

    Las funciones de gestión predefinidas utilizan programas de utilidad de gestión incorporados de JMX. Para habilitar las funciones de gestión predefinidas, es necesario definir varias propiedades del sistema Java. Debe definir como mínimo la propiedad siguiente:

    `com.sun.management.jmxremote.port=`*`número_puerto`*

    Además, debe comprobar que la autentificación y el protocolo SSL estén configurados debidamente.

    Para obtener información sobre la habilitación de las funciones de gestión predefinidas, consulte el sitio Web situado en este URL:

`http://download.oracle.com/javase/6/docs/technotes/guides/management/agent.html`

  - Escriba un agente JMX. Esta técnica se describe también en:

`http://download.oracle.com/javase/6/docs/technotes/guides/management/agent.html`

    En el ejemplo siguiente, se crea un conector RMI para PlatformMBeanServer utilizando el objeto `MyCustomJMXAuthenticator`. La clase `MyCustomJMXAuthenticator` define cómo las credenciales remotas se convierten en un sujeto JAAS implementando la interfaz JMXAuthenticator:

```
...
HashMap<String> env = new HashMap<String>();
env.put(JMXConnectorServer.AUTHENTICATOR, new MyCustomJMXAuthenticator());
env.put("jmx.remote.x.access.file", "my.access.file");

MBeanServer mbs =
  java.lang.management.ManagementFactory.getPlatformMBeanServer();
JMXServiceURL url =
  new JMXServiceURL("service:jmx:rmi:///jndi/rmi://:9999/jmxrmi");

JMXConnectorServer cs =
  JMXConnectorServerFactory.newJMXConnectorServer(url, env, mbs);
cs.start();
...
public class MyCustomJMXAuthenticator implements JMXAuthenticator {

  public Subject authenticate(Object credentials) {
    // hash contiene el nombre de usuario, contraseña, etc...
    Hashtable <String> credentialsHash
      = (Hashtable <String>) credentials;

    ...
    // Autentificar utilizando credenciales proporcionadas
    ...
    if (authentication-successful) {
      return new Subject(true,
        Collections.singleton
          (new JMXPrincipal(credentialsHash.get("username"))),
        Collections.EMPTY_SET,
        Collections.EMPTY_SET);
    }
    throw new SecurityException("Invalid credentials");
  }
}
```

Acceso al controlador de rastreo remoto

Puede acceder al controlador de rastreo remoto mediante herramientas de gestión proporcionadas o a través de una aplicación.

Acerca de esta tarea

Utilice las herramientas de gestión proporcionadas a través de un cliente de gestión compatible con JMX, tal como JConsole, que forma parte de Java Standard Edition, Versión 6. Encontrará información sobre la utilización de JConsole como herramienta de gestión en este URL:

`http://download.oracle.com/javase/6/docs/technotes/guides/management/jconsole.html`

En una aplicación que accede al controlador de rastreo remoto, el controlador de rastreo remoto es un bean gestionado (MBean). JMX gestiona recursos a través de agentes JMX. Un agente JMX es un servidor MBean. Cada MBean representa un recurso. Cada MBean tiene un nombre, que el usuario define mediante un objeto de clase javax.management.ObjectName. Utilice el objeto ObjectName para registrar y recuperar MBeans en el servidor MBean.

El nombre de MBean consta de dos partes: el dominio y las propiedades de clave. Para el ObjectName del controlador de rastreo remoto de IBM Data Server Driver para JDBC y SQLJ, el dominio es com.ibm.db2.jcc, y las propiedades de clave son name=DB2TraceManager.

Una aplicación que accede al controlador de rastreo remoto debe incluir estos pasos:

Procedimiento

1. Establecer una conexión de Invocación de método remoto (RMI) con un servidor MBean.
2. Realizar una búsqueda en el controlador de rastreo remoto del servidor MBean.
3. Invocar operaciones de rastreo en el MBean.

   Puede trabajar en el MBean de las maneras siguientes:
   - Utilizando un proxy MBean
   - Sin un proxy, a través de MBeanServerConnection.

Ejemplo

Ejemplo: acceso al controlador de rastreo remoto sin proxies: Este ejemplo muestra el acceso a MBeans directamente desde MBeanServerConnection. Este método es el más genérico pues no necesita definiciones de interfaz apropiadas en la aplicación cliente JMX.

```
Hashtable<String> env = new Hashtable<String>();
env.put(Context.INITIAL_CONTEXT_FACTORY,
 "com.sun.jndi.fscontext.RefFSContextFactory");

try {
  System.out.println ("");
  System.out.println ("-------------------------------------------------");
  System.out.println ("Establecer conexión RMI con un MBeanServer");
  System.out.println ("-------------------------------------------------");
  JMXServiceURL url =
    new JMXServiceURL ("service:jmx:rmi:///jndi/rmi://localhost:9999/jmxrmi");
  JMXConnector jmxc = JMXConnectorFactory.connect (url, env);
  MBeanServerConnection mbsc = jmxc.getMBeanServerConnection();
```

```
  System.out.println ("");
  System.out.println ("-----------------------------------------------");
  System.out.println ("Se está procesando MBean");
  System.out.println ("-----------------------------------------------");
  String objectNameString = "com.ibm.db2.jcc:name=DB2TraceManager";
  ObjectName name = new ObjectName(objectNameString);
  System.out.println ("ObjectName="+objectNameString);

  System.out.println ("");
  System.out.println ("-----------------------------------------------");
  System.out.println ("Mostrar todos los atributos del MBean");
  System.out.println ("-----------------------------------------------");

  System.out.println(
    "TraceDirectory  = "+mbsc.getAttribute (name, "TraceDirectory"));
  System.out.println(
    "TraceFile       = "+mbsc.getAttribute (name, "TraceFile"));
  System.out.println(
    "TraceFileAppend = "+mbsc.getAttribute (name, "TraceFileAppend"));
  System.out.println(
    "TraceLevel      = "+mbsc.getAttribute (name, "TraceLevel"));

  System.out.println ("");
  System.out.println ("-----------------------------------------------");
  System.out.println ("Invocar algunas operaciones en el MBean");
  System.out.println ("-----------------------------------------------");
  System.out.print ("Se está invocando suspendTrace()...");
  mbsc.invoke (name, "suspendTrace", null , null);
  System.out.println ("éxito");

  System.out.print ("Se está invocando resumeTrace()...");
  mbsc.invoke (name, "resumeTrace", null , null);
  System.out.println ("éxito");
}
catch (Exception e) {
  System.out.println ("error");
  e.printStackTrace();
}
```

Ejemplo: acceso al controlador de rastreo remoto con proxies: Este ejemplo muestra la creación de un proxy para un MBean. El proxy implementa la interfaz com.ibm.db2.jcc.mx.DB2TraceManagerMXBean. La aplicación realiza llamadas directamente en el proxy, y la implementación subyacente del proxy invoca la actuación del MBean en el servidor remoto MBean.

```
Hashtable<String> env = new Hashtable<String>();
env.put(Context.INITIAL_CONTEXT_FACTORY,
 "com.sun.jndi.fscontext.RefFSContextFactory");

try {
  System.out.println ("");
  System.out.println ("-----------------------------------------------");
  System.out.println ("Establecer conexión RMI con un MBeanServer");
  System.out.println ("-----------------------------------------------");
  JMXServiceURL url =
    new JMXServiceURL ("service:jmx:rmi:///jndi/rmi://localhost:9999/jmxrmi");
  JMXConnector jmxc = JMXConnectorFactory.connect (url, env);
  MBeanServerConnection mbsc = jmxc.getMBeanServerConnection();

  System.out.println ("");
  System.out.println ("-----------------------------------------------");
  System.out.println ("Se está procesando MBean");
  System.out.println ("-----------------------------------------------");
  String objectNameString = "com.ibm.db2.jcc:name=DB2TraceManager";
  ObjectName name = new ObjectName(objectNameString);
  System.out.println ("ObjectName="+objectNameString);
```

```
  System.out.println ("");
  System.out.println ("-------------------------------------------------");
  System.out.println ("Mostrar todos los atributos del MBean");
  System.out.println ("-------------------------------------------------");
  com.ibm.db2.jcc.mx.DB2TraceManagerMXBean mbeanProxy =
    JMX.newMBeanProxy(mbsc, name,
      com.ibm.db2.jcc.mx.DB2TraceManagerMXBean.class, true);
  System.out.println ("TraceDirectory  = "+mbeanProxy.getTraceDirectory ());
  System.out.println ("TraceFile       = "+mbeanProxy.getTraceFile ());
  System.out.println ("TraceFileAppend = "+mbeanProxy.getTraceFileAppend ());
  System.out.println ("TraceLevel      = "+mbeanProxy.getTraceLevel ());
  System.out.println ("");
  System.out.println ("-------------------------------------------------");
  System.out.println ("Invocar algunas operaciones en el MBean");
  System.out.println ("-------------------------------------------------");
  System.out.print ("Se está invocando suspendTrace()...");
  mbeanProxy.suspendTrace();
  System.out.println ("éxito");
  System.out.print ("Se está invocando resumeTrace()...");
  mbeanProxy.resumeTrace();
  System.out.println ("éxito");
}
catch (Exception e) {
  System.out.println ("error");
  e.printStackTrace();
}
```

Capítulo 9. Soporte de cliente de Java para obtener alta disponibilidad en los servidores de datos de IBM

Las aplicaciones cliente que se conectan con DB2 Database para Linux, UNIX y Windows, DB2 para z/OS o con IBM Informix pueden beneficiarse fácilmente de las características de alta disponibilidad de esos servidores de datos.

Las aplicaciones cliente pueden usar las siguientes funciones de alta disponibilidad:

- Redireccionamiento de cliente automático

  La función de redireccionamiento de cliente automático está disponible en todos los servidores de datos de IBM. El redireccionamiento de cliente automático utiliza información proporcionada por los servidores de datos para redirigir aplicaciones cliente de un servidor que experimente una parada a un servidor alternativo. El redireccionamiento automático de cliente permite que las aplicaciones continúen con su trabajo con una interrupción mínima. La redirección del trabajo a un servidor alternativo recibe el nombre de *migración tras error*.

  Para conexiones con servidores de datos DB2 para z/OS, el redireccionamiento de cliente automático forma parte de la función de equilibrado de la carga de trabajo. Por lo general, para el servidor DB2 para z/OS, no se debe habilitar el redireccionamiento de cliente automático sin el equilibrado de la carga de trabajo.
- Afinidades de cliente

  Afinidades de cliente es una solución de migración tras error que el cliente controla completamente. Está pensada para situaciones donde debe conectarse con un servidor primario determinado. Si se produce una parada durante la conexión con el servidor primario, utilice las afinidades de cliente para imponer un orden específico para la migración tras error a servidores alternativos.

  Las afinidades de cliente no son aplicables a un entorno de compartimiento de datos de DB2 para z/OS, porque todos los miembros de un grupo de compartimiento de datos puede acceder a los datos de forma simultánea. El compartimiento de datos es la solución que se recomienda para la alta disponibilidad para DB2 para z/OS.
- Equilibrado de la carga de trabajo

  El equilibrado de la carga de trabajo está disponible en todos los servidores de datos de IBM. El equilibrado de la carga de trabajo garantiza que el trabajo se distribuya de forma eficaz entre los servidores de un clúster de alta disponibilidad de IBM Informix, un grupo de compartimiento de datos de DB2 para z/OS o una instancia DB2 pureScale de DB2 Database para Linux, UNIX y Windows.

La tabla siguiente ofrece enlaces a la información del servidor sobre estas funciones.

Tabla 35. Información del servidor sobre alta disponibilidad

Servidor de datos	Temas relacionados
DB2 Database para Linux, UNIX y Windows	• DB2 pureScale: Guía básica para la documentación de característica DB2 pureScale • Redireccionamiento automático de clientes: Guía del redireccionamiento automático de clientes
IBM Informix	Gestión de las conexiones de clúster con el gestor de conexiones
DB2 para z/OS	Comunicación con los grupos de compartición de datos

Importante: Para obtener información sobre las conexiones con DB2 para z/OS, esta información trata las conexiones directas con DB2 para z/OS. Para obtener información sobre la alta disponibilidad para conexiones a través del servidor de DB2 Connect, consulte la documentación de DB2 Connect.

Soporte de cliente de Java a fin de obtener alta disponibilidad para conexiones con servidores DB2 Database para Linux, UNIX y Windows

Los servidores DB2 Database para Linux, UNIX y Windows ofrecen alta disponibilidad para aplicaciones cliente, mediante el equilibrado de la carga de trabajo y el redireccionamiento automático de clientes. Este soporte está disponible para aplicaciones que usan clientes de Java (JDBC, SQLJ o pureQuery), así como clientes que no son de Java (ODBC, CLI, .NET, OLE DB, PHP, Ruby o SQL incorporado).

Para los clientes de Java, debe utilizar IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 para beneficiarse del soporte de alta disponibilidad de DB2 Database para Linux, UNIX y Windows. Necesita IBM Data Server Driver para JDBC y SQLJ versión 3.58 o 4.8, o una versión posterior.

El soporte de alta disponibilidad para conexiones con servidores DB2 Database para Linux, UNIX y Windows incluye:

Redireccionamiento de cliente automático
Este soporte permite a un cliente recuperarse de una anomalía intentando reconectarse a la base de datos a través de un servidor alternativo. La reconexión con otro servidor recibe el nombre de *migración tras error*. Para los clientes de Java, siempre está habilitado el soporte de redireccionamiento automático de clientes.

Los servidores pueden proporcionar la función de redireccionamiento automático de clientes de las formas siguientes:

- Hay varios servidores configurados en una instancia DB2 pureScale. Una conexión con una base de datos es una conexión con un miembro de dicha instancia DB2 pureScale. La migración tras error implica la reconexión con otro miembro de la instancia DB2 pureScale. Este entorno requiere que los clientes utilicen TCP/IP para conectar con la instancia DB2 pureScale.
- Hay una instancia DB2 pureScale y un servidor alternativo definidos para una base de datos. La migración tras error implica en primer lugar la reconexión con otro miembro de la instancia DB2 pureScale. La migración tras error al servidor alternativo se intentará únicamente si no hay disponible ningún miembro de la instancia DB2 pureScale.

- Hay una instancia DB2 pureScale definida para el servidor primario y otra instancia DB2 pureScale definida para el servidor alternativo. La migración tras error implica en primer lugar la reconexión con otro miembro de la instancia DB2 pureScale primaria. La migración tras error a la instancia DB2 pureScale alternativa se intentará únicamente si no hay disponible ningún miembro de la instancia DB2 pureScale primaria.
- Una base de datos está definida en un único servidor. La configuración de dicha base de datos incluye la especificación de un servidor alternativo. La migración tras error implica la reconexión con el servidor alternativo.

Para las aplicaciones cliente de Java, la migración tras error para el redireccionamiento automático de clientes puede ser *sin fisuras* o *con fisuras*. Mediante la migración tras error sin fisuras, cuando la aplicación cliente se reconecta a otro servidor, siempre se devuelve un error a la aplicación, para indicar que se ha producido la migración tras error (conexión con el servidor alternativo). En la migración tras error sin fisuras, el controlador no devuelve un error si el error de conexión y la reconexión con un servidor alternativo se producen durante la ejecución de la primera sentencia de SQL de una transacción.

En una instancia DB2 pureScale, se puede utilizar el soporte de redireccionamiento automático de cliente con o sin equilibrado de la carga de trabajo.

Equilibrado de la carga de trabajo

El equilibrado de la carga de trabajo puede mejorar la disponibilidad de una instancia DB2 pureScale.

Con el equilibrado de la carga de trabajo, una instancia DB2 pureScale garantiza que el trabajo se distribuye de forma eficaz entre los miembros.

Los clientes de Java en cualquier sistema operativo dan soporte al equilibrado de la carga. La conexión desde el cliente hasta la instancia DB2 pureScale debe utilizar TCP/IP.

Cuando el equilibrado de la carga de trabajo está habilitado, el cliente recibe con frecuencia información del estado de los miembros de la instancia DB2 pureScale a través de una lista de servidores. El cliente coloca la lista de servidores en antememoria y utiliza la información que contiene para determinar el miembro al que debería redireccionarse la siguiente transacción.

Para las aplicaciones Java, cuando se utiliza JNDI, varias JVM pueden compartir la lista de servidores colocada en antememoria para la primera conexión. No obstante, el equilibrado de la carga de trabajo siempre se realiza en el contexto de una única JVM.

DB2 Database para Linux, UNIX y Windows soporta dos tipos de equilibrados de la carga de trabajo:

Equilibrado de la carga de trabajo en el nivel de conexión
El equilibrado de la carga de trabajo en el nivel de conexión se realiza en los límites de las conexiones. No está soportado para los clientes de Java.

Equilibrado de la carga de trabajo en el nivel de transacción
El equilibrado de la carga de trabajo en el nivel de transacción se realiza en los límites de las transacciones. El soporte de cliente para el equilibrado de la carga de trabajo en el nivel de transacción está inhabilitado por omisión para los clientes que se conectan con DB2 Database para Linux, UNIX y Windows.

Afinidades de cliente
Afinidades de cliente es una solución de redireccionamiento de cliente automático que el cliente controla completamente. Está pensada para situaciones donde debe conectarse con un servidor primario determinado. Si se produce una parada durante la conexión con el servidor primario, utilice las afinidades de cliente para imponer un orden específico para la migración tras error a servidores alternativos.

Configuración del soporte de redireccionamiento automático de clientes de DB2 Database para Linux, UNIX y Windows para clientes de Java

Para las conexiones con bases de datos DB2 Database para Linux, UNIX y Windows, el proceso para configurar el soporte de redireccionamiento automático de clientes en los clientes de Java es el mismo que para las conexiones con un entorno que no es un entorno DB2 pureScale y un entorno DB2 pureScale.

El soporte de redireccionamiento automático de clientes para aplicaciones cliente de Java que se conectan a DB2 Database para Linux, UNIX y Windows funciona para las conexiones obtenidas mediante la interfaz javax.sql.DataSource, javax.sql.ConnectionPoolDataSource, `javax.sql.XADataSource` o java.sql.DriverManager.

Para configurar el redireccionamiento de cliente automático en un cliente de IBM Data Server Driver para JDBC y SQLJ:

1. Defina las propiedades adecuadas para especificar las direcciones del servidor primario y del servidor alternativo que utilizar si se produce una anomalía en la primera conexión.
   - Si la aplicación está utilizando la interfaz DriverManager para conexiones:
     a. Especifique el nombre de servidor y número de puerto del servidor primario que desee utilizar en el URL de conexión.
     b. Utilice el nombre de servidor y número de puerto del servidor alternativo que desee utilizar como valor de las propiedades clientRerouteAlternateServerName y clientRerouteAlternatePortNumber.

     Restricción: El soporte del redireccionamiento de cliente automático para conexiones establecidas mediante la interfaz DriverManager tiene estas restricciones:
     – La información sobre el servidor alternativo se puede compartir entre conexiones DriverManager solamente si las conexiones se crean con el mismo URL y las mismas propiedades.
     – La propiedad clientRerouteServerListJNDIName o las propiedades clientRerouteServerListJNDIContext no se pueden definir para una conexión DriverManager.
     – El redireccionamiento de cliente automático no está habilitado para las conexiones por omisión (jdbc:default:connection).
   - Si la aplicación está utilizando la interfaz DataSource para conexiones, utilice una de las técnicas siguientes o las dos:
     – Defina los nombres de servidor y números de puerto en las propiedades de DataSource:
       a. Utilice el nombre de servidor y número de puerto del servidor primario que desee utilizar como valor de las propiedades serverName y portNumber.

b. Utilice el nombre de servidor y número de puerto del servidor alternativo que desee utilizar como valor de las propiedades clientRerouteAlternateServerName y clientRerouteAlternatePortNumber.

– Configure JNDI para el redireccionamiento automático de clientes utilizando una instancia de DB2ClientRerouteServerList para identificar el servidor primario y el servidor alternativo.

a. Cree una instancia de DB2ClientRerouteServerList.

DB2ClientRerouteServerList es un bean de Java serializable con las propiedades siguientes:

Nombre de propiedad	Tipo de datos
com.ibm.db2.jcc.DB2ClientRerouteServerList.alternateServerName	String[]
com.ibm.db2.jcc.DB2ClientRerouteServerList.alternatePortNumber	int[]
com.ibm.db2.jcc.DB2ClientRerouteServerList.primaryServerName	String[]
com.ibm.db2.jcc.DB2ClientRerouteServerList.primaryPortNumber	int[]

Para cada propiedad se definen los métodos getXXX y setXXX.

b. Utilice el nombre de servidor y número de puerto del servidor primario que desee utilizar como valor de las propiedades com.ibm.db2.jcc.DB2ClientRerouteServerList.primaryServerName y com.ibm.db2.jcc.DB2ClientRerouteServerList.primaryPortNumber.

c. Utilice los nombres de servidor y los números de puerto del servidor alternativo que desee utilizar como valor de las propiedades com.ibm.db2.jcc.DB2ClientRerouteServerList.alternateServerName y com.ibm.db2.jcc.DB2ClientRerouteServerList.alternatePortNumber.

d. Para hacer que DB2ClientRerouteServerList sea persistente:

1) Vincule la instancia de DB2ClientRerouteServerList con el registro de JNDI.

2) Asigne el nombre de JNDI del objeto DB2ClientRerouteServerList a la propiedad clientRerouteServerListJNDIName de IBM Data Server Driver para JDBC y SQLJ.

3) Asigne el nombre del contexto JNDI que se utiliza para la vinculación y la consulta de la instancia DB2ClientRerouteServerList en la propiedad clientRerouteServerListJNDIContext.

Cuando una fuente de datos DataSource se configura para utilizar JNDI a fin de almacenar información alternativa sobre el redireccionamiento automático del cliente, las propiedades estándar del servidor y puerto de la fuente de datos DataSource no se utilizan para una petición getConnection. En su lugar, se obtiene la dirección del servidor primario a partir de la información transitoria de clientRerouteServerList. Si el almacén de datos JNDI no está disponible debido a un error de vinculación o búsqueda de JNDI, IBM Data Server Driver para JDBC y SQLJ intenta crear una conexión utilizando las propiedades estándar de servidor y puerto de DataSource. Se producen avisos para indicar que se ha producido un error de vinculación o búsqueda de JNDI.

Después de una migración tras error:

- IBM Data Server Driver para JDBC y SQLJ intenta propagar la información actualizada del servidor al almacén de datos JNDI.

- Los valores primaryServerName y primaryPortNumber que se especifican en DB2ClientRerouteServerList se utilizan para la conexión. Si no se especifica primaryServerName, se utilizarán los valores serverName y portNumber de la instancia de DataSource.

Si configura las propiedades de DataSource, así como la configuración de JNDI, para el redireccionamiento automático de clientes, las propiedades de DataSource prevalecen sobre la configuración de JNDI.

2. Defina propiedades para controlar el número de reintentos, el tiempo entre los reintentos y la frecuencia con la que se actualiza la lista de servidores.

   Las siguientes propiedades controlan el comportamiento de los reintentos para el redireccionamiento de cliente automático.

   maxRetriesForClientReroute
   Número máximo de reintentos de conexión para el redireccionamiento de cliente automático.

   Cuando no está configurado el soporte de afinidades de cliente, si no se ha definido maxRetriesForClientReroute o retryIntervalForClientReroute, el comportamiento por omisión consiste en que la conexión se reintente durante 10 minutos, con un tiempo de espera entre reintentos que aumenta a medida que se incrementa el tiempo transcurrido desde el primer reintento.

   Cuando están configuradas las afinidades de cliente, el valor por omisión de maxRetriesForClientReroute es 3.

   retryIntervalForClientReroute
   Número de segundos entre reintentos de conexión consecutivos.

   Cuando no está configurado el soporte de afinidades de cliente, si no se ha definido retryIntervalForClientReroute o maxRetriesForClientReroute, el comportamiento por omisión consiste en que la conexión se reintente durante 10 minutos, con un tiempo de espera entre reintentos que aumenta a medida que se incrementa el tiempo transcurrido desde el primer reintento.

   Cuando están configuradas las afinidades de cliente, el valor por omisión de retryIntervalForClientReroute es 0 (sin espera).

Ejemplo de habilitación del soporte de redireccionamiento automático de clientes de DB2 Database para Linux, UNIX y Windows en aplicaciones Java

La configuración del cliente de Java para el soporte de redireccionamiento automático de clientes de DB2 Database para Linux, UNIX y Windows incluye la configuración de varias propiedades de IBM Data Server Driver para JDBC y SQLJ.

El ejemplo siguiente muestra la configuración de aplicaciones cliente de Java para el soporte de redireccionamiento automático de clientes de DB2 Database para Linux, UNIX y Windows.

Suponga que la instalación contiene un servidor primario y un servidor alternativo que tienen los nombres de servidor y números de puerto siguientes:

Nombre de servidor	Número de puerto
srv1.sj.ibm.com	50000
srv3.sj.ibm.com	50002

El código siguiente define propiedades de DataSource en una aplicación para que la aplicación se conecte a srv1.sj.ibm.com como servidor primario, y a srv3.sj.ibm.com como servidor alternativo. Es decir, si srv1.sj.ibm.com está inactivo durante la conexión inicial, el controlador debe conectar con srv3.sj.ibm.com.

```
ds.setDriverType(4);
ds.setServerName("srv1.sj.ibm.com");
ds.setPortNumber("50000");
ds.setClientRerouteAlternateServerName("srv3.sj.ibm.com");
ds.setClientRerouteAlternatePortNumber("50002");
```

El código siguiente configura JNDI para el redireccionamiento de cliente automático. El código crea una instancia de DB2ClientRerouteServerList, vincula esa instancia con el registro de JNDI, y asigna el nombre JNDI del objeto DB2ClientRerouteServerList a la propiedad clientRerouteServerListJNDIName.

```
// Crear un contexto inicial para operaciones de asignación de nombres
InitialContext registry = new InitialContext();
// Crear un objeto DB2ClientRerouteServerList
DB2ClientRerouteServerList address = new DB2ClientRerouteServerList();

// Definir el número de puerto y nombre de servidor para el servidor principal
address.setPrimaryPortNumber(50000);
address.setPrimaryServerName("srv1.sj.ibm.com");

// Definir el número de puerto y nombre de servidor para el servidor alternativo
int[] port = {50002};
String[] server = {"srv3.sj.ibm.com"};
address.setAlternatePortNumber(port);
address.setAlternateServerName(server);

registry.rebind("serverList", address);
// Asignar el nombre JNDI del objeto DB2ClientRerouteServerList a
// la propiedad clientRerouteServerListJNDIName
datasource.setClientRerouteServerListJNDIName("serverList");
```

Configuración del soporte de equilibrado de la carga de trabajo de DB2 Database para Linux, UNIX y Windows para clientes de Java

Para configurar una aplicación cliente de IBM Data Server Driver para JDBC y SQLJ que se conecte con una instancia DB2 pureScale de DB2 Database para Linux, UNIX y Windows para el equilibrado de la carga de trabajo, tiene que conectar con un miembro de la instancia DB2 pureScale y establecer las propiedades que habilitan el equilibrado de la carga de trabajo y el número máximo de conexiones.

Las aplicaciones cliente de Java dan soporte al equilibrado de la carga de trabajo en el nivel de transacción. No soportan el equilibrado de la carga de trabajo en el nivel de conexión. El equilibrado de la carga de trabajo se soporta únicamente para conexiones con una instancia DB2 pureScale.

El soporte de equilibrado de la carga de trabajo para aplicaciones cliente de Java que se conectan con DB2 Database para Linux, UNIX y Windows funciona para conexiones que se obtienen mediante la interfaz javax.sql.DataSource, javax.sql.ConnectionPoolDataSource, `javax.sql.XADataSource` o java.sql.DriverManager.

Restricción: La utilización del equilibrado de la carga de trabajo para conexiones establecidas mediante la interfaz DriverManager tiene estas restricciones:

- La información sobre el servidor alternativo se puede compartir entre conexiones DriverManager solamente si las conexiones se crean con el mismo URL y las mismas propiedades.
- La propiedad clientRerouteServerListJNDIName o las propiedades clientRerouteServerListJNDIContext no se pueden definir para una conexión DriverManager.
- El equilibrado de la carga de trabajo no está habilitado para las conexiones por omisión (jdbc:default:connection).

La tabla siguiente describe los valores básicos de las propiedades para habilitar el equilibrado de la carga de trabajo de DB2 Database para Linux, UNIX y Windows para aplicaciones Java.

Tabla 36. Valores básicos para habilitar el soporte del equilibrado de la carga de trabajo en aplicaciones Java

Valor de IBM Data Server Driver para JDBC y SQLJ	Valor
Propiedad enableSysplexWLB	true
Propiedad maxTransportObjects	Número máximo de conexiones que puede establecer el peticionario con la instancia DB2 pureScale
Dirección de conexión: servidor	Dirección IP de un miembro de una instancia DB2 pureScale[1]
Dirección de conexión: puerto	Número de puerto SQL para la instancia DB2 pureScale[1]
Dirección de conexión: base de datos	Nombre de la base de datos

Nota:

1. Alternativamente, puede utilizar un distribuidor, como Websphere Application Server Network Deployment, o un DNS con múltiples direcciones ("multihoming") para establecer la conexión inicial con la base de datos.
   - Para un distribuidor, hay que especificar el puerto y la dirección IP del distribuidor. El distribuidor analiza la distribución actual de la carga de trabajo y utiliza esta información para reenviar la petición de conexión a uno de los miembros de la instancia DB2 pureScale.
   - Para el DNS con múltiples direcciones ("multihoming"), debe especificar una dirección IP y un número de puerto que se puedan resolver en la dirección IP y el número de puerto de cualquier miembro de la instancia DB2 pureScale. El proceso del DNS con múltiples direcciones selecciona un miembro basándose en unos criterios concretos, como la selección rotativa simple o la distribución de la carga de trabajo por miembro.

Si desea ajustar el soporte del equilibrado de la carga de trabajo de DB2 Database para Linux, UNIX y Windows, existen propiedades de configuración globales. Las propiedades para IBM Data Server Driver para JDBC y SQLJ se enumeran en la tabla siguiente.

Tabla 37. Propiedades de configuración para ajustar el soporte del equilibrado de la carga de trabajo de DB2 Database para Linux, UNIX y Windows para conexiones desde IBM Data Server Driver para JDBC y SQLJ

Propiedad de configuración de IBM Data Server Driver para JDBC y SQLJ	Descripción
db2.jcc.maxRefreshInterval	Especifica la cantidad máxima de tiempo en segundos entre las distintas actualizaciones de la copia de cliente de la lista de servidores que se utiliza para el equilibrado de la carga de trabajo. El valor por omisión es 10. El valor mínimo válido es 1.
db2.jcc.maxTransportObjectIdleTime	Especifica el tiempo máximo transcurrido en número de segundos antes de descartar un transporte desocupado. El valor por omisión es 10. El valor mínimo soportado es 0.
db2.jcc.maxTransportObjectWaitTime	Especifica el número de segundos que el cliente deberá esperar a que esté disponible un transporte. El valor por omisión es 1. El valor mínimo soportado es 0.
db2.jcc.minTransportObjects	Especifica el límite inferior del número de objetos de transporte de una agrupación de objetos de transporte global. El valor por omisión es 0. Cualquier valor que sea menor o igual que 0 significa que la agrupación de objetos de transporte global puede llegar a estar vacía.

Ejemplo de habilitación del soporte de equilibrado de la carga de trabajo de DB2 Database para Linux, UNIX y Windows en aplicaciones Java

La configuración del cliente de Java para el soporte del equilibrado de la carga de trabajo de DB2 Database para Linux, UNIX y Windows incluye la configuración de varias propiedades de IBM Data Server Driver para JDBC y SQLJ.

El ejemplo siguiente muestra la configuración de aplicaciones cliente de Java para el soporte de equilibrado de la carga de trabajo de DB2 Database para Linux, UNIX y Windows.

Antes de configurar el cliente, se deben configurar los servidores a los que se conecta el cliente en una instancia DB2 pureScale.

Siga estos pasos para configurar el cliente:

1. Compruebe que el controlador IBM Data Server Driver para JDBC y SQLJ esté al nivel correcto para utilizar el equilibrado de la carga de trabajo. Para ello siga estos pasos:
   a. Emita el mandato siguiente en una ventana de línea de mandatos:
      ```
      java com.ibm.db2.jcc.DB2Jcc -version
      ```
   b. Busque una línea como la siguiente en los datos de salida y compruebe que *nnn* sea 3.58 o posterior.
   c.
      ```
      [jcc] Driver: IBM Data Server Driver for JDBC and SQLJ Architecture nnn xxx
      ```
2. Establezca las propiedades de IBM Data Server Driver para JDBC y SQLJ para habilitar el concentrador de conexiones o el equilibrado de carga:
   a. Establezca estas propiedades de Connection o DataSource:
      - enableSysplexWLB
      - maxTransportObjects
   b. Establezca la propiedad de configuración global db2.jcc.maxRefreshInterval en un archivo DB2JccConfiguration.properties para establecer el intervalo

máximo de renovación para todas las instancias de DataSource o Connection que se crean en el controlador.

Empiece con una configuración similar a la siguiente:

Tabla 38. Ejemplo de valores de propiedad para el equilibrado de carga de DB2 Database para Linux, UNIX y Windows

Propiedad	Valor
enableSysplexWLB	true
maxTransportObjects	80
db2.jcc.maxRefreshInterval	10

Los valores que se especifican no se dan como recomendación. Los valores deben determinarse basándose en factores como el número de conexiones físicas disponibles. El número de objetos de transporte debe ser igual o mayor que el número de objetos de conexión.

3. Para ajustar el equilibrado de carga de todas las instancias de DataSource o Connection que se crean en el controlador, establezca la propiedad de configuración db2.jcc.maxTransportObjects en un archivo DB2JccConfiguration.properties.

   Comience con un valor similar al siguiente:

   ```
   db2.jcc.maxTransportObjects=1000
   ```

4. **Opcional:** Especifique nombres de servidores alternativos en las propiedades clientRerouteAlternateServername y clientRerouteAlternatePortNumber. Este paso no es necesario para habilitar el equilibrado de la carga de trabajo. Sin embargo, es útil especificar una lista de servidores alternativos para garantizar que la primera conexión podrá establecerse correctamente en caso de que el servidor primario no esté disponible.

Funcionamiento del redireccionamiento automático de clientes para las conexiones con DB2 Database para Linux, UNIX y Windows desde clientes de Java

Cuando está habilitado el soporte de redireccionamiento de clientes de IBM Data Server Driver para JDBC y SQLJ, una aplicación Java que esté conectada a una base de datos DB2 Database para Linux, UNIX y Windows puede seguir trabajando cuando el servidor primario tiene una anomalía.

El redireccionamiento automático de clientes para una aplicación Java que está conectada con una base de datos DB2 Database para Linux, UNIX y Windows actúa de la manera siguiente cuando está inhabilitado el soporte de afinidades de cliente:

1. Durante cada conexión a la fuente de datos, IBM Data Server Driver para JDBC y SQLJ obtiene información sobre los servidores primarios e intercambiables.
   - Para la primera conexión con una base de datos DB2 Database para Linux, UNIX y Windows:
     a. Si se establecen las propiedades clientRerouteAlternateServerName y clientRerouteAlternatePortNumber, IBM Data Server Driver para JDBC y SQLJ carga dichos valores en memoria como valores de servidor alternativo, junto con los valores de servidor primario serverName y portNumber.
     b. Si no se establecen las propiedades clientRerouteAlternateServerName y clientRerouteAlternatePortNumber y se configura un almacén de JNDI estableciendo la propiedad clientRerouteServerListJNDIName en

DB2BaseDataSource, IBM Data Server Driver para JDBC y SQLJ carga la información de servidor primario y alternativo desde el almacén de JDNI en memoria.

c. Si no hay ninguna propiedad DataSource establecida para los servidores alternativos, y JNDI no está configurado, IBM Data Server Driver para JDBC y SQLJ examina tablas DNS para encontrar información del servidor primario y alternativo. Si existe información de DNS, IBM Data Server Driver para JDBC y SQLJ carga esos valores en la memoria.

   En un entorno DB2 pureScale, independientemente del resultado de la búsqueda de DNS:

   1) si la propiedad de configuración db2.jcc.outputDirectory está establecida, IBM Data Server Driver para JDBC y SQLJ busca en el directorio especificado en db2.jcc.outputDirectory un archivo llamado jccServerListCache.bin.
   2) Si db2.jcc.outputDirectory no se ha establecido y la propiedad del sistema java.io.tmpdir está establecida, IBM Data Server Driver para JDBC y SQLJ busca en el directorio especificado en java.io.tmpdir un archivo llamado jccServerListCache.bin.
   3) Si se puede acceder a jccServerListCache.bin, IBM Data Server Driver para JDBC y SQLJ carga la antememoria en la memoria y obtiene la información del servidor alternativo de jccServerListCache.bin correspondiente al valor de serverName definido para el objeto DataSource.

d. Si no está disponible ninguna información del servidor primario o alternativo, no se puede establecer una conexión e IBM Data Server Driver para JDBC y SQLJ emite una excepción.

- Para conexiones subsiguientes, IBM Data Server Driver para JDBC y SQLJ obtiene valores de los servidores primarios e intercambiables desde la memoria del controlador.

2. IBM Data Server Driver para JDBC y SQLJ intenta conectarse a la fuente de datos utilizando el nombre y el número de puerto del servidor primario.

   Cuando no se trata de un entorno DB2 pureScale, el servidor primario es un servidor autónomo. Cuando se trata de un entorno DB2 pureScale, el servidor primario es un miembro de una instancia DB2 pureScale.

   Si la conexión se realiza a través de la interfaz DriverManager, IBM Data Server Driver para JDBC y SQLJ crea un objeto DataSource interno para el proceso del redireccionamiento automático de clientes.

3. Si falla la conexión con el servidor primario:

   a. Si se trata de la primera conexión, IBM Data Server Driver para JDBC y SQLJ intenta volverse a conectar con un servidor utilizando la información proporcionada por las propiedades de controlador, como clientRerouteAlternateServerName y clientRerouteAlternatePortNumber.
   b. Si no se trata de la primera conexión, IBM Data Server Driver para JDBC y SQLJ intenta establecer una conexión mediante la información de la lista de servidores más reciente que se recibe del servidor.

   La conexión con un servidor alternativo se denomina *migración tras error*.

   IBM Data Server Driver para JDBC y SQLJ utiliza las propiedades maxRetriesForClientReroute y retryIntervalForClientReroute para determinar cuántas veces se debe reintentar la conexión y cuánto tiempo se debe esperar entre los reintentos. El intento de conectar con el servidor primario y los servidores alternativos se contabiliza como un solo reintento.

4. Si la conexión no se establece, no están definidos maxRetriesForClientReroute y retryIntervalForClientReroute y los valores originales de serverName y portNumber que están definidos en DataSource son distintos de los utilizados para la conexión actual, la conexión se recupera con los valores serverName y portNumber que están definidos en DataSource.
5. Si la migración tras error es satisfactoria durante la conexión inicial, el controlador genera un SQLWarning. Si se produce una migración tras error satisfactoria después de la conexión inicial:
   - Si se ha habilitado la migración tras error sin fisuras y se satisfacen las condiciones siguientes, el controlador intenta de nuevo la transacción en el nuevo servidor, sin avisar a la aplicación.
     - La propiedad enableSeamlessFailover está establecida en DB2BaseDataSource.YES (1).
     - La conexión no está en la transacción. Es decir, la anomalía se produce cuando la primera sentencia de SQL se ejecuta en la transacción.
     - No existen tablas temporales globales activas en el servidor.
     - No hay cursores retenidos abiertos.
   - Si la migración tras error sin fisuras no está vigente, el controlador emite una SQLException a la aplicación con el código de error -4498, lo que indica a la aplicación que la conexión se ha restablecido automáticamente y que la transacción se ha retrotraído de forma implícita. A continuación, la aplicación puede intentar su transacción sin realizar una retroacción en primer lugar.

     Un código de razón devuelto con el código de error -4498 indica si los registros especiales del servidor de bases de datos que se han modificado durante la conexión original se restablecerán en la conexión de migración tras error.

   Puede determinar si se ha utilizado la información de servidor alternativo al establecer la conexión inicial llamando al método DB2Connection.alternateWasUsedOnConnect.
6. Tras la migración tras error, la memoria del controlador se actualiza con la información del nuevo servidor primario y alternativo que se devuelve desde el servidor primario nuevo.

Ejemplos

Ejemplo: redireccionamiento de cliente automático a un servidor DB2 Database para Linux, UNIX y Windows si todavía no se han definido maxRetriesForClientReroute ni retryIntervalForClientReroute: Suponga que se han definido las propiedades siguientes para una conexión con una base de datos:

Propiedad	Valor
enableClientAffinitiesList	DB2BaseDataSource.NO (2)
serverName	host1
portNumber	port1
clientRerouteAlternateServerName	host2
clientRerouteAlternatePortNumber	port2

Los pasos siguientes muestran un caso de redireccionamiento de cliente automático para una conexión con un servidor DB2 Database para Linux, UNIX y Windows:

1. IBM Data Server Driver para JDBC y SQLJ carga host1:port1 en la memoria como dirección del servidor primario, y carga host2:port2 en la memoria como dirección del servidor alternativo.
2. Durante la conexión inicial, el controlador intenta conectar con host1:port1.
3. La conexión con host1:port1 falla, por lo que el controlador intenta otra conexión con host1:port1.
4. La reconexión con host1:port1 falla, por lo que el controlador intenta conectar con host2:port2.
5. La conexión con host2:port2 se realiza satisfactoriamente.
6. El controlador recupera información sobre el servidor alternativo recibida del servidor host2:port2, y actualiza su memoria con esa información.

   Supongamos que el controlador recibe una lista de servidores en la que aparecen host2:port2, host2a:port2a. host2:port2 se almacena como servidor primario nuevo, y host2a:port2a se almacena como servidor alternativo nuevo. Si se detecta otro error de comunicación en esta misma conexión, o en otra conexión creada a partir de la misma DataSource, el controlador intenta conectar con host2:port2 en calidad de nuevo servidor primario. Si esa conexión falla, el controlador intenta conectar con el nuevo servidor alternativo host2a:port2a.
7. Se produce un error de comunicación durante la conexión con host2:port2.
8. El controlador intenta conectar con host2a:port2a.
9. La conexión con host2a:port2a es satisfactoria.
10. El controlador recupera información sobre el servidor alternativo recibida del servidor host2a:port2a, y actualiza su memoria con esa información.

Ejemplo: redireccionamiento de cliente automático a un servidor DB2 Database para Linux, UNIX y Windows en un entorno DB2 pureScale, si todavía no se han definido maxRetriesForClientReroute ni retryIntervalForClientReroute y la propiedad de configuración db2.jcc.outputDirectory está establecida: Suponga que se han definido las propiedades siguientes para una conexión que se establece desde DataSource A:

Propiedad	**Valor**
enableClientAffinitiesList	DB2BaseDataSource.NO (2)
serverName	host1
portNumber	port1
db2.jcc.outputDirectory (propiedad de configuración)	/home/tmp

Los pasos siguientes muestran un caso de redireccionamiento de cliente automático para una conexión con un servidor DB2 Database para Linux, UNIX y Windows:

1. Con la información de DataSource A, IBM Data Server Driver para JDBC y SQLJ carga host1:port1 en la memoria como dirección del servidor primario. El controlador busca el archivo de antememoria jccServerListCache.bin en /home/tmp, pero no existe.
2. La conexión con host1:port1 se realiza satisfactoriamente. Suponga que el servidor devuelve una lista de servidores que contiene host1:port1 y host2:port2.
3. El controlador crea una antememoria en la memoria, con una entrada que especifica host2:port2 como lista de servidores alternativos para host1:port1. A

continuación, el controlador crea el archivo de antememoria /home/tmp/jccServerListCache.bin, y graba en este archivo la antememoria de la memoria.

4. La conexión de la aplicación A con host1:port1 falla, por lo que el controlador intenta conectar con host2:port2.
5. La conexión de la aplicación A con host2:port2 se realiza satisfactoriamente. Supongamos que el servidor devuelve una lista de servidores que contiene host2:port2 y host2a:port2a. host2:port2 es el servidor primario nuevo y host2a:port2a es el servidor alternativo nuevo.
6. El controlador busca información de servidor alternativo para host2:port2 en la antememoria almacenada en la memoria, pero no la encuentra. Crea una entrada nueva en la antememoria almacenada en la memoria para host2:port2, con host2a:port2a como lista de servidores alternativos. El controlador actualiza el archivo de antememoria /home/tmp/jccServerListCache.bin con la entrada nueva que se ha añadido a la antememoria almacenada en la memoria.
7. La aplicación A finaliza y la JVM se cierra.
8. La aplicación B, que también utiliza DataSource A, se inicia.
9. El controlador carga la lista de servidores del archivo de antememoria /home/tmp/jccServerListCache.bin y localiza la entrada para host1:port1, que especifica host2:port2 como lista de servidores alternativos. El controlador establece host2:port2 como lista de servidores alternativos para host1:port1.
10. Se produce un error en las comunicaciones cuando la aplicación B intenta conectar con host1:port1.
11. La aplicación B intenta conectar con el servidor alternativo host2:port2.
12. La conexión con host2:port2 se realiza satisfactoriamente. La aplicación B continúa.

Ejemplo: redireccionamiento de cliente automático hacia un servidor DB2 Database para Linux, UNIX y Windows cuando las propiedades maxRetriesForClientReroute y retryIntervalForClientReroute están definidas para varios reintentos: Suponga que están definidas las propiedades siguientes para una conexión con una base de datos:

Propiedad	Valor
enableClientAffinitiesList	DB2BaseDataSource.NO (2)
serverName	host1
portNumber	port1
clientRerouteAlternateServerName	host2
clientRerouteAlternatePortNumber	port2
maxRetriesForClientReroute	3
retryIntervalForClientReroute	2

Los pasos siguientes muestran un caso de redireccionamiento de cliente automático para una conexión con un servidor DB2 Database para Linux, UNIX y Windows:

1. IBM Data Server Driver para JDBC y SQLJ carga host1:port1 en la memoria como dirección del servidor primario, y carga host2:port2 en la memoria como dirección del servidor alternativo.
2. Durante la conexión inicial, el controlador intenta conectar con host1:port1.
3. La conexión con host1:port1 falla, por lo que el controlador intenta otra conexión con host1:port1.

4. La conexión con host1:port1 falla de nuevo, por lo que el controlador intenta conectar con host2:port2.
5. La conexión con host2:port2 falla.
6. El controlador espera dos segundos.
7. El controlador intenta conectar con host1:port1 y falla.
8. El controlador intenta conectar con host2:port2 y falla.
9. El controlador espera dos segundos.
10. El controlador intenta conectar con host1:port1 y falla.
11. El controlador intenta conectar con host2:port2 y falla.
12. El controlador espera dos segundos.
13. El controlador emite una excepción de SQL (SQLException) que incluye el código de error -4499.

Funcionamiento del soporte de grupos alternativos

El soporte de grupos alternativos permite que IBM Data Server Driver para JDBC y SQLJ mueva una carga de trabajo de aplicaciones a una instancia DB2 pureScale alternativa cuando la instancia DB2 pureScale primaria no está disponible.

Importante: Si está utilizando la versión de IBM Data Server Driver para JDBC y SQLJ que se proporciona con DB2 9.7 Fixpack 6, debe aplicar el APAR IC79084 para que el soporte de grupos alternativos esté disponible en el sistema.

El soporte de grupos alternativos se habilita proporcionando las direcciones de las instancias DB2 pureScale alternativas en las propiedades alternateGroupServerName, alternateGroupPortNumber y alternateGroupDatabaseName de Connection o DataSource.

Asimismo, puede controlar si el comportamiento de la migración tras error sin fisuras ha de estar o no en vigor para el soporte de grupos alternativos estableciendo la propiedad enableAlternateGroupSeamlessACR de Connection o DataSource.

Tras una migración tras error desde el grupo primario hasta un grupo alternativo, el valor de la propiedad databaseName no cambia.

Para que el funcionamiento del soporte de grupos alternativos sea correcto, los datos del grupo primario y de los grupos alternativos deben ser los mismos.

La migración tras error sin fisuras de grupo alternativo permite realizar la migración tras error sin fisuras únicamente desde el grupo primaria a un grupo alternativo. Después de la migración tras error sin fisuras, todas las conexiones en una instancia de DataSource se establecen con el grupo alternativo. DataSource no puede crear conexiones de vuelta al grupo primario, aunque el grupo primario pase a estar disponible y todas las conexiones existentes con el grupo alternativo se hayan cerrado. Después de que las conexiones en una instancia de DataSource se hayan movido al grupo alternativo, la única forma de asociar estas conexiones con el grupo primario es reciclando el entorno de ejecución de Java (JVM). Si una instancia de DataSource se está ejecutando dentro de Websphere Application Server, se debe reciclar el servidor de aplicaciones completo para mover las conexiones al grupo primario.

Después de la migración tras error sin fisuras, si una nueva instancia de DataSource se instancia mediante una aplicación dentro de la misma JVM desde la

que anteriormente las conexiones han realizado una migración tras error sin fisuras a un servidor alternativo, IBM Data Server Driver para JDBC y SQLJ permite establecer conexiones con el grupo primario cuando el grupo primario pase a estar disponible, incluso si otras conexiones de DataSource que se ejecutan dentro de la misma JVM debe conectar con el grupo alternativo.

Si una conexión que creó mediante DriverManager.getConnection realiza una migración tras error sin fisuras a un grupo alternativo, todas las conexiones posteriores que se obtienen a través de DriverManager.getConnection y tienen el mismo URL y las mismas propiedades también se conectan al grupo alternativo, incluso si el grupo primario pasa a estar disponible. La única forma de mover una conexión al grupo primario con DriverManager.getConnection es creando una conexión con URL o propiedades diferentes.

El soporte de grupos alternativos funciona del modo siguiente:

- Para la primera conexión de una aplicación con una instancia DB2 pureScale primaria:
  1. IBM Data Server Driver para JDBC y SQLJ intenta conectar la aplicación con la instancia DB2 pureScale primaria.
  2. Si la conexión falla, el controlador intenta conectar la aplicación con la instancia DB2 pureScale alternativa que se especifica mediante el primer conjunto de valores de las propiedades alternateGroupServerName, alternateGroupPortNumber y alternateGroupDatabaseName.
  3. Si la conexión falla, el controlador intenta conectar la aplicación con la instancia DB2 pureScale alternativa que se especifica mediante el siguiente conjunto de valores de las propiedades alternateGroupServerName, alternateGroupPortNumber y alternateGroupDatabaseName. El controlador sigue con este paso hasta que se ha establecido una conexión satisfactoriamente o el controlador ha probado todos los conjuntos de valores de las propiedades alternateGroupServerName, alternateGroupPortNumber y alternateGroupDatabaseName.
  4. Si no se establece una conexión, el controlador devuelve un error de SQL -4499 a la aplicación.
- Para una conexión posterior, tras haberse conectado la aplicación con la instancia DB2 pureScale primaria:
  1. IBM Data Server Driver para JDBC y SQLJ intenta conectar la aplicación con cada uno de los miembros disponibles de la instancia DB2 pureScale primaria.
  2. Si ninguno de los miembros de la instancia DB2 pureScale primaria está disponible al primer intento, el controlador vuelve a intentar la conexión con la instancia DB2 pureScale primaria utilizando la dirección que especifica el conjunto de valores de las propiedades serverName, portNumber y databaseName de Connection o DataSource.
  3. Si la conexión con la instancia DB2 pureScale primaria falla, el controlador intenta conectar la aplicación con la instancia DB2 pureScale alternativa que se especifica mediante el primer conjunto de valores de las propiedades alternateGroupServerName, alternateGroupPortNumber y alternateGroupDatabaseName.
  4. Si la conexión falla, el controlador intenta conectar la aplicación con la instancia DB2 pureScale alternativa que se especifica mediante el siguiente conjunto de valores de las propiedades alternateGroupServerName, alternateGroupPortNumber y alternateGroupDatabaseName. El controlador sigue con este paso hasta que se ha establecido una conexión

satisfactoriamente o el controlador ha probado todos los conjuntos de valores de las propiedades alternateGroupServerName, alternateGroupPortNumber y alternateGroupDatabaseName.

5. Si no se establece una conexión, el controlador devuelve un error de SQL -4499 a la aplicación.

- Para una conexión posterior, tras haberse conectado la aplicación con la instancia DB2 pureScale alternativa:
  1. Si ninguno de los miembros de la instancia DB2 pureScale alternativa está disponible, el controlador vuelve a intentar la conexión con la misma instancia DB2 pureScale alternativa utilizando la dirección que se especifique para ese grupo en las propiedades alternateGroupServerName, alternateGroupPortNumber y alternateGroupDatabaseName de Connection o DataSource.
  2. Si la conexión con la misma instancia DB2 pureScale alternativa falla, el controlador intenta conectar la aplicación con la instancia DB2 pureScale que se especifica mediante el siguiente conjunto de entradas en alternateGroupServerName, alternateGroupPortNumber y alternateGroupDatabaseName.
  3. Si la conexión falla, el controlador intenta conectar la aplicación con la instancia DB2 pureScale alternativa que se especifica mediante el siguiente conjunto de valores de las propiedades alternateGroupServerName, alternateGroupPortNumber y alternateGroupDatabaseName. El controlador sigue con este paso hasta que se ha establecido una conexión satisfactoriamente o el controlador ha probado todos los conjuntos de valores de las propiedades alternateGroupServerName, alternateGroupPortNumber y alternateGroupDatabaseName.
  4. Si no se establece una conexión, el controlador devuelve un error de SQL -4499 a la aplicación.
- Para una conexión con un grupo primario que se encuentra en una transacción:
  1. IBM Data Server Driver para JDBC y SQLJ intenta conectar la aplicación con una instancia DB2 pureScale alternativa.
  2. Si se establece una conexión, enableAlternateSeamlessGroupACR se ha establecido en `true` y la transacción es apta para la migración tras error sin fisuras, vuelve a intentarse la transacción.
  3. Si se establece una conexión, enableAlternateSeamlessGroupACR se ha establecido en `true` y la transacción no es apta para la migración tras error sin fisuras, el controlador devuelve el error de SQL -30108 a la aplicación.
  4. Si se establece una conexión y enableAlternateSeamlessGroupACR se ha establecido en `false`, el controlador devuelve el error de SQL -30108 a la aplicación.
  5. Si no se establece una conexión, el controlador devuelve un error de SQL -4499 a la aplicación.

Ejemplos

Supongamos que se han definido tres grupos: PG1, AG1 y AG2. PG1 es el grupo primario y AG1 y AG2 son los grupos alternativos para el soporte de grupos alternativos de IBM Data Server Driver para JDBC y SQLJ.

Supongamos que los grupos tienen los valores de servidor, puerto y base de datos siguientes:

Grupo	Valores de servidor, puerto y base de datos
PG1	host1, port1, dbname1
AG1	host2, port2, dbname2
AG2	host3, port3, dbname3

Supongamos también que se han establecido los valores de propiedad siguientes:

Propiedad	Valor
serverName	host1
portNumber	port1
databaseName	dbname1
alternateGroupServerName	host2,host3
alternateGroupPortNumber	port2,port3
alternateGroupDatabaseName	dbname2,dbname3
enableAlternateGroupSeamlessACR	true

En los pasos siguientes se ofrece una demostración de una situación de ejemplo de grupo alternativo para una conexión con PG1 que falla:

1. El controlador intenta conectar la aplicación con PG1 utilizando host1:port1.
2. La conexión falla.
3. El controlador intenta conectar la aplicación con AG1 utilizando host2:port2.
4. La conexión se establece satisfactoriamente.
5. La aplicación sigue ejecutándose.
6. Todos los miembros de AG1 dejan de estar disponibles y la conexión con AG1 falla.
7. El controlador intenta conectar la aplicación con AG1 utilizando host2:port2.
8. La conexión falla.
9. El controlador intenta conectar la aplicación con AG2 utilizando host3:port3.
10. La conexión falla.
11. El controlador emite el error de SQL -4499.

En los pasos siguientes se ofrece una demostración de una situación de ejemplo de grupo alternativo para una conexión con PG1 que no se ejecuta correctamente durante una transacción:

1. El controlador intenta conectar la aplicación con PG1 utilizando host1:port1.
2. La conexión es correcta.
3. La aplicación inicia la ejecución del trabajo.
4. Todos los miembros de PG1 caen.
5. El controlador intenta conectar la aplicación con AG1 utilizando host2:port2.
6. La conexión se establece satisfactoriamente.
7. La aplicación cumple los criterios para la migración tras error sin fisuras, por lo tanto, vuelve a intentarse la transacción.
8. El reintento no se ejecuta correctamente.
9. El controlador emite el error de SQL -30108 y retrotrae el trabajo hasta el punto de confirmación anterior.

Funcionamiento del equilibrado de la carga de trabajo para conexiones con DB2 Database para Linux, UNIX y Windows

El equilibrado de la carga de trabajo, denominado también equilibrado de la carga de trabajo en el nivel de transacción, para conexiones con DB2 Database para Linux, UNIX y Windows contribuye a la alta disponibilidad al equilibrar el trabajo entre los servidores de una instancia DB2 pureScale en el comienzo de una transacción.

En la visión general siguiente se describen los pasos que tienen lugar cuando un cliente se conecta a una instancia DB2 pureScale de DB2 Database para Linux, UNIX y Windows y se habilita el equilibrado de la carga de trabajo en el nivel de transacción:

1. Cuando el cliente establece la primera conexión con la instancia DB2 pureScale, el miembro al que se conecta el cliente devuelve una lista de servidores con los detalles de conexión (dirección IP, puerto y peso) para los miembros de la instancia DB2 pureScale.

   El cliente guarda la lista de servidores en la memoria caché. La vida útil por omisión de la lista de servidores en la memoria caché es de 30 segundos.
2. Al inicio de una transacción nueva, el cliente lee la lista de servidores colocada en antememoria para identificar a un servidor que tenga capacidad sin utilizar y busca en la agrupación de transporte un transporte desocupado que esté vinculado al servidor infrautilizado. (Un transporte desocupado es un transporte que no tiene ningún objeto de conexión asociado).
   - Si un transporte desocupado está disponible, el cliente asocia el objeto de conexión con el transporte.
   - Si, después de un tiempo de espera excedido configurable por el usuario (db2.jcc.maxTransportObjectWaitTime para un cliente de Java o maxTransportWaitTime para un cliente que no es de Java), no hay ningún transporte desocupado disponible en la agrupación de transporte y no puede asignarse ningún transporte nuevo puesto que la agrupación de transporte ha alcanzado su límite, se devuelve un error a la aplicación.
3. Cuando la transacción se ejecuta, accede al servidor que está vinculado al transporte.

   Cuando la primera sentencia de SQL en una transacción se ejecuta, si IBM Data Server Driver para JDBC y SQLJ recibe un error de comunicación porque el servidor de datos desactiva la conexión o se sobrepasa el valor blockingReadConnectionTimeout, el controlador vuelve a intentar la sentencia de SQL 10 veces antes de notificar un error. En cada reintento, el controlador cierra el transporte existente, obtiene un nuevo transporte y, a continuación, ejecuta la transacción. Durante estos reintentos, si las propiedades maxRetriesForClientReroute y retryIntervalForClientReroute están establecidas, sus valores sólo se aplican al proceso de obtener un nuevo transporte durante cada reintento.
4. Cuando la transacción finaliza, el cliente verifica con el servidor que la reutilización del transporte aún está permitida para el objeto de conexión.
5. Si la reutilización del transporte aún está permitida, el servidor devolverá una lista de sentencias SET para registros especiales que se apliquen al entorno de ejecución para el objeto de conexión.

   El cliente almacena en la antememoria estas sentencias, que reproduce con el fin de reconstruir el entorno de ejecución cuando el objeto de conexión está asociado con un transporte nuevo.

6. El objeto de la conexión se desasocia entonces del transporte, si el cliente determina que necesita hacerlo.
7. La copia de cliente de la lista de servidores se renueva cuando se establece una nueva conexión, cada 30 segundos o según un intervalo configurado por el usuario.
8. Cuando el equilibrado de la carga de trabajo en el nivel de transacción es obligatorio para una transacción nueva, el cliente utiliza el proceso descrito anteriormente para asociar el objeto de conexión con un transporte.

Requisitos de programación de aplicaciones para obtener una alta disponibilidad para conexiones con servidores DB2 Database para Linux, UNIX y Windows

La migración tras error para el redireccionamiento de cliente automático puede ser con o sin fisuras. Si la migración tras error para las conexiones con DB2 Database para Linux, UNIX y Windows no se produce sin fisuras, debe agregar el código para tener en cuenta los errores que se han devuelto cuando se produce la migración tras error.

Si una migración tras error no se produce sin fisuras y se vuelve a establecer una conexión con el servidor, SQLCODE -4498 (para clientes de Java) o SQL30108N (para clientes que no son de Java) se devuelve a la aplicación. Todo el trabajo que se ha producido en la transacción actual se ha retrotraído. En la aplicación, necesita:

- Compruebe el código de razón que se devuelve con el error. Determine si se transfieren los valores de registros especiales en el miembro de compartimiento de datos que falla al nuevo miembro de compartimiento de datos (migración tras error). Restaure cualquier valor de registro especial que no sea actual.
- Ejecute todas las operaciones de SQL que han tenido lugar durante la transacción anterior.

Deben cumplirse las condiciones siguientes para la migración tras error para que las conexiones con DB2 Database para Linux, UNIX y Windows sean sin fisuras:

- El lenguaje de programación de la aplicación es Java, CLI o .NET.
- La conexión no está en la transacción. Es decir, la anomalía se produce cuando la primera sentencia de SQL se ejecuta en la transacción.
- Si está habilitado el equilibrado de la carga en el nivel de transacción, el servidor de datos permite la reutilización del transporte al final de la transacción anterior.
- Todos los datos de sesiones globales se han cerrado o descartado.
- No hay cursores retenidos abiertos.
- Si la aplicación utiliza CLI, ésta no podrá llevar a cabo acciones que requieran al controlador que mantenga un historial de las API a las que se ha llamado anteriormente, con el fin de reproducir la sentencia de SQL. Ejemplos de dichas acciones son la especificación de datos durante la ejecución, ejecutar sentencias de SQL compuesto o utilizar entradas de matrices.
- La aplicación no es un procedimiento almacenado.
- Autocommit no se ha habilitado. La migración tras error sin fisuras puede producirse cuando se habilita Autocommit. Sin embargo, la situación siguiente puede producir problemas: suponga que la tarea de SQL se ejecuta satisfactoriamente y se confirma en el servidor de datos, pero la conexión o el servidor queda inactivo antes de que el reconocimiento de la operación de confirmación se vuelva a enviar al cliente. Cuando el cliente restablece la

conexión, vuelve a reproducir la sentencia de SQL confirmada anteriormente. El resultado es que la sentencia de SQL se ejecuta dos veces. Para evitar esta situación, desactive la confirmación automática cuando habilite la migración tras error sin fisuras.

Afinidades de cliente para DB2 Database para Linux, UNIX y Windows

Afinidades de cliente es un método sólo para clientes para proporcionar funciones de redireccionamiento de cliente automático.

Hay afinidades de cliente disponibles para aplicaciones que utilicen CLI, .NET o Java (IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4). El redireccionamiento está controlado por el controlador.

Las afinidades de cliente están pensadas para situaciones donde debe conectarse con un servidor primario determinado. Si se produce una parada durante la conexión con el servidor primario, debe imponer un orden específico para la migración tras error a servidores alternativos. Debe utilizar afinidades de cliente para el redireccionamiento de cliente automático, sólo si el redireccionamiento de cliente automático que utiliza las funciones de migración tras error del servidor no funciona en su entorno.

Como parte de la configuración de afinidades de cliente, especifique una lista de servidores alternativos, así como el orden en que se probarán las conexiones con los servidores alternativos. Cuando las afinidades de cliente estén en uso, las conexiones se establecen de acuerdo con la lista de servidores alternativos, en lugar del nombre de host y el número de puerto especificados por la aplicación. Por ejemplo, si una aplicación especifica que se ha establecido una conexión con server1, pero el proceso de configuración especifica que los servidores deben probarse en el orden (server2, server3, server1), la conexión inicial se establece con server2 en lugar de con server1.

La migración tras error con afinidades de cliente será sin fisuras, si se dan las condiciones siguientes:

- La conexión no está en la transacción. Es decir, la anomalía se produce cuando la primera sentencia de SQL se ejecuta en la transacción.
- No existen tablas temporales globales activas en el servidor.
- No hay cursores retenidos abiertos.

Si utiliza afinidades de cliente, puede especificar que si el servidor primario vuelve a estar operativo después de una parada, las conexiones vuelven de un servidor alternativo al servidor primario en un límite de transacción. Esta actividad recibe el nombre de *recuperación*.

Configuración de afinidades de cliente de Java para conexiones de DB2 Database para Linux, UNIX y Windows

Para habilitar el soporte de afinidades de cliente en aplicaciones Java, defina las propiedades para indicar que desea usar las afinidades de cliente y para especificar los servidores primarios y alternativos.

La tabla siguiente describe los valores de las propiedades para habilitar afinidades de cliente para aplicaciones Java.

Tabla 39. Valores de propiedades para habilitar afinidades de cliente para aplicaciones Java

Valor de IBM Data Server Driver para JDBC y SQLJ	Valor
enableClientAffinitiesList	DB2BaseDataSource.YES (1)
clientRerouteAlternateServerName	Una lista separada por comas de los servidores primarios y alternativos
clientRerouteAlternatePortNumber	Una lista separada por comas de los números de puertos de los servidores primarios y alternativos
enableSeamlessFailover	DB2BaseDataSource.YES (1) para migración tras error sin fisuras; DB2BaseDataSource.NO (2) o enableSeamlessFailover sin especificar para ninguna migración tras error sin fisuras
maxRetriesForClientReroute	Número de veces que se volverá a intentar la conexión con cada servidor, incluido el servidor primario, después de que falle una conexión con el servidor primario. El valor por omisión es 3.
retryIntervalForClientReroute	Número de segundos que se debe esperar entre reintentos. El valor por omisión es sin espera.
affinityFailbackInterval	Número de segundos que se debe esperar después de que el primer límite de transacción se transfiera al servidor primario. Defina este valor si desea realizar la transferencia al servidor primario.

Ejemplo de habilitación de afinidades en clientes de Java para conexiones de DB2 Database para Linux, UNIX y Windows

Para utilizar las afinidades de cliente en el redireccionamiento de cliente automático en aplicaciones Java, debe definir propiedades para indicar que desea utilizar afinidades de cliente, así como para identificar los servidores alternativos y primarios.

El ejemplo siguiente muestra cómo habilitar afinidades de cliente para una migración tras error sin transferencia.

Suponga que establece las propiedades siguientes para una conexión con una base de datos:

Propiedad	Valor
enableClientAffinitiesList	DB2BaseDataSource.YES (1)
clientRerouteAlternateServername	host1,host2,host3
clientRerouteAlternatePortNumber	port1,port2,port3
maxRetriesForClientReroute	3
retryIntervalForClientReroute	2

Suponga que se produce un error de comunicación durante una conexión con el servidor identificado por host1:port1. Los pasos siguientes muestran el redireccionamiento de cliente automático con afinidades de cliente.

1. El controlador intenta conectar con host1:port1.

2. La conexión con host1:port1 falla.
3. El controlador espera dos segundos.
4. El controlador intenta conectar con host1:port1.
5. La conexión con host1:port1 falla.
6. El controlador espera dos segundos.
7. El controlador intenta conectar con host1:port1.
8. La conexión con host1:port1 falla.
9. El controlador espera dos segundos.
10. El controlador intenta conectar con host2:port2.
11. La conexión con host2:port2 falla.
12. El controlador espera dos segundos.
13. El controlador intenta conectar con host2:port2.
14. La conexión con host2:port2 falla.
15. El controlador espera dos segundos.
16. El controlador intenta conectar con host2:port2.
17. La conexión con host2:port2 falla.
18. El controlador espera dos segundos.
19. El controlador intenta conectar con host3:port3.
20. La conexión con host3:port3 falla.
21. El controlador espera dos segundos.
22. El controlador intenta conectar con host3:port3.
23. La conexión con host3:port3 falla.
24. El controlador espera dos segundos.
25. El controlador intenta conectar con host3:port3.
26. La conexión con host3:port3 falla.
27. El controlador espera dos segundos.
28. El controlador emite una excepción de SQL (SQLException) que incluye el código de error -4499.

El ejemplo siguiente muestra cómo habilitar afinidades de cliente para una migración tras error con transferencia.

Suponga que establece las propiedades siguientes para una conexión con una base de datos:

Propiedad	Valor
enableClientAffinitiesList	DB2BaseDataSource.YES (1)
clientRerouteAlternateServername	host1,host2,host3
clientRerouteAlternatePortNumber	port1,port2,port3
maxRetriesForClientReroute	3
retryIntervalForClientReroute	2
affinityFailbackInterval	300

Suponga que el administrador de la base de datos retira el servidor identificado por host1:port1 a mantenimiento, después de que se establezca una conexión con

host1:port1. Los pasos siguientes muestran la migración tras error a un servidor alternativo y la retrotracción al servidor primario después de que se haya completado el mantenimiento.

1. El controlador se conecta satisfactoriamente con host1:port1 en nombre de una aplicación.
2. El administrador de la base de datos desactiva host1:port1.
3. La aplicación intenta realizar tareas en la conexión.
4. El controlador transfiere satisfactoriamente el control a host2:port2.
5. Cuando han transcurrido 200 segundos, la tarea se confirma.
6. Cuando han transcurrido 300 segundos, ha transcurrido el intervalo de recuperación. El controlador comprueba si el servidor primario está activo. No está activo, así que no se produce la recuperación.
7. Cuando han transcurrido 350 segundos, host1:port1 vuelve a quedar en línea.
8. La aplicación continúa realizando tareas en host2:port2, porque no ha transcurrido el intervalo de recuperación más reciente.
9. Cuando han transcurrido 600 segundos, ha vuelto a transcurrir el intervalo de recuperación. El controlador comprueba si el servidor primario está activo. Ahora está activo.
10. Cuando han transcurrido 650 segundos, la tarea se confirma.
11. Cuando han transcurrido 651 segundos, la aplicación intenta iniciar una nueva transacción en host2:port2. Se produce la recuperación en el puerto host1:port1; por lo tanto, la nueva transacción se inicia en host1:port1.

Soporte de cliente de Java a fin de obtener alta disponibilidad para conexiones con servidores IBM Informix

El soporte de clústeres de alta disponibilidad en servidores IBM Informix proporciona alta disponibilidad a aplicaciones cliente, a través del equilibrado de la carga de trabajo y el redireccionamiento de cliente automático. Este soporte está disponible para aplicaciones que usan clientes de Java (JDBC, SQLJ o pureQuery), o bien clientes que no son de Java (ODBC, CLI, .NET, OLE DB, PHP, Ruby o SQL incorporado).

Para los clientes de Java, debe utilizar IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 para beneficiarse del soporte de clústeres para alta disponibilidad de IBM Informix.

Para los clientes que no son de Java, debe utilizar uno de los clientes o paquetes de clientes siguientes para beneficiarse del soporte de clústeres de alta disponibilidad:

- IBM Data Server Client
- IBM Data Server Runtime Client
- IBM Data Server Driver Package
- IBM Data Server Driver para ODBC y CLI

El soporte de clústeres para alta disponibilidad para conexiones con servidores de IBM Informix incluye:

Redireccionamiento de cliente automático
Este soporte permite a un cliente recuperarse de una anomalía intentando reconectarse a la base de datos a través de un servidor disponible en un clúster de alta disponibilidad. La reconexión con otro servidor recibe el nombre de

migración tras error. Habilite el redireccionamiento de cliente automático en el cliente mediante la habilitación en éste del equilibrado de la carga de trabajo.

En un entorno de IBM Informix, los servidores primario y de reserva corresponden a miembros de un clúster de alta disponibilidad que es controlado por un gestor de conexiones. Si hay varios gestores de conexiones, cliente puede usarlos para determinar la información de los servidores primarios y alternativos. El cliente utiliza los gestores de conexión alternativa sólo para la conexión inicial.

La migración tras error para el redireccionamiento de cliente automático puede ser *sin fisuras* o *con fisuras*. Mediante la migración tras error sin fisuras, cuando la aplicación cliente se reconecta a un servidor alternativo, éste siempre devuelve un error a la aplicación, para indicar que se ha producido la migración tras error (conexión con el servidor alternativo).

Para las aplicaciones cliente de Java, CLI o .NET, la migración tras error para el redireccionamiento de cliente automático puede ser sin fisuras o con fisuras. La migración tras error sin fisuras significa que cuando la aplicación se reconecta satisfactoriamente con un servidor alternativo, éste no devuelve un error a la aplicación.

Equilibrado de la carga de trabajo
El equilibrado de la carga de trabajo puede mejorar la disponibilidad de un clúster de alta disponibilidad de IBM Informix. Cuando el equilibrado de la carga de trabajo está habilitado, el cliente obtiene regularmente información de estado sobre los miembros de un clúster de alta disponibilidad. El cliente utiliza esta información para determinar el servidor al que se debe encaminar la transacción siguiente. Gracias al equilibrado de la carga de trabajo, los gestores de conexiones de IBM Informix garantizan que el trabajo se distribuya de forma eficiente entre los servidores, y que dicho trabajo se transfiera a otro servidor si un servidor presenta una anomalía.

Concentrador de conexiones
Este soporte está disponible para aplicaciones Java que se conectan con IBM Informix. El concentrador de conexiones reduce los recursos necesarios en los servidores de bases de datos IBM Informix para dar soporte a un gran número de estaciones de trabajo y usuarios web. Con el concentrador de conexiones, sólo se necesitan algunas conexiones físicas activas simultáneas para dar soporte a varias aplicaciones que acceden al servidor de bases de datos de forma simultánea. Cuando se habilita el equilibrado de la carga de trabajo en un cliente de Java, se habilita automáticamente el concentrador de conexiones.

Afinidades de cliente
Afinidades de cliente es una solución de redireccionamiento de cliente automático que el cliente controla completamente. Está pensada para situaciones donde debe conectarse con un servidor primario determinado. Si se produce una parada durante la conexión con el servidor primario, utilice las afinidades de cliente para imponer un orden específico para la migración tras error a servidores alternativos.

Configuración del soporte de alta disponibilidad de IBM Informix para clientes de Java

Para configurar una aplicación cliente de IBM Data Server Driver para JDBC y SQLJ que se conecta con un clúster de alta disponibilidad de IBM Informix, debe conectarse con una dirección que represente un gestor de conexiones y establecer las propiedades que habilitan el equilibrado de la carga de trabajo y el número máximo de conexiones.

El soporte de alta disponibilidad para clientes de Java que se conectan con IBM Informix funciona para conexiones que se obtienen mediante la interfaz javax.sql.DataSource, javax.sql.ConnectionPoolDataSource, `javax.sql.XADataSource` o java.sql.DriverManager.

Restricción: El soporte de alta disponibilidad para conexiones establecidas mediante la interfaz DriverManager tiene estas restricciones:

- La información sobre el servidor alternativo se puede compartir entre conexiones DriverManager solamente si las conexiones se crean con el mismo URL y las mismas propiedades.
- La propiedad clientRerouteServerListJNDIName o las propiedades clientRerouteServerListJNDIContext no se pueden definir para una conexión DriverManager.
- El soporte de alta disponibilidad no está habilitado para las conexiones por omisión (jdbc:default:connection).

Para poder habilitar IBM Data Server Driver para JDBC y SQLJ a fin de obtener una alta disponibilidad para conexiones con IBM Informix, la instalación debe tener uno o más gestores de conexiones, un servidor primario y uno o más servidores alternativos.

La tabla siguiente describe los valores básicos de las propiedades para habilitar el equilibrado de la carga de trabajo para aplicaciones Java.

Tabla 40. Valores básicos para habilitar el soporte de alta disponibilidad de IBM Informix en aplicaciones Java

Valor de IBM Data Server Driver para JDBC y SQLJ	Valor
Propiedad enableSysplexWLB	true
Propiedad maxTransportObjects	Número máximo de conexiones que el peticionario puede realizar al clúster de alta disponibilidad
Dirección de conexión: servidor	Dirección IP de un gestor de conexiones. Consulte la "Configuración de las propiedades del puerto y del servidor para conectarse con un gestor de conexiones" en la página 291.
Dirección de conexión: puerto	Número de puerto SQL para el gestor de conexiones. Consulte la "Configuración de las propiedades del puerto y del servidor para conectarse con un gestor de conexiones" en la página 291.
Dirección de conexión: base de datos	Nombre de la base de datos

Si desea habilitar el concentrador de conexión, pero no desea habilitar el equilibrado de la carga de trabajo, puede usar estas propiedades.

Tabla 41. Configuraciones para habilitar el concentrador de conexión de IBM Informix sin el equilibrado de la carga de trabajo en las aplicaciones Java

Valor de IBM Data Server Driver para JDBC y SQLJ	Valor
Propiedad enableSysplexWLB	false
Propiedad enableConnectionConcentrator	true

Si desea refinar el soporte de alta disponibilidad de IBM Informix, existen propiedades adicionales. Las propiedades para IBM Data Server Driver para JDBC y SQLJ se enumeran en la tabla siguiente. Estas propiedades son propiedades de configuración y no propiedades de Connection o DataSource.

Tabla 42. Propiedades para refinar el soporte de alta disponibilidad de IBM Informix para conexiones desde IBM Data Server Driver para JDBC y SQLJ

Propiedad de configuración de IBM Data Server Driver para JDBC y SQLJ	Descripción
db2.jcc.maxTransportObjectIdleTime	Especifica el tiempo máximo transcurrido en número de segundos antes de descartar un transporte desocupado. El valor por omisión es 10. El valor mínimo soportado es 0.
db2.jcc.maxTransportObjectWaitTime	Especifica el número de segundos que el cliente deberá esperar a que esté disponible un transporte. El valor por omisión es 1. El valor mínimo soportado es 0.
db2.jcc.minTransportObjects	Especifica el límite inferior del número de objetos de transporte de una agrupación de objetos de transporte global. El valor por omisión es 0. Cualquier valor que sea menor o igual que 0 significa que la agrupación de objetos de transporte global puede llegar a estar vacía.

Configuración de las propiedades del puerto y del servidor para conectarse con un gestor de conexiones

Para establecer el número de puerto y el servidor a fin de conectarse con un gestor de conexiones, siga este proceso:

- Si el clúster de alta disponibilidad está utilizando un solo gestor de conexiones, y la aplicación está utilizando la interfaz DataSource para conexiones, utilice el nombre de servidor y el número de puerto del gestor de conexiones como valores de las propiedades serverName y portNumber.
- Si el clúster de alta disponibilidad está utilizando un solo gestor de conexiones, y la aplicación está utilizando la interfaz DriverManager para conexiones, especifique el nombre de servidor y el número de puerto del gestor de conexiones en el URL de conexión.
- Si el clúster de alta disponibilidad está utilizando más de un gestor de conexiones, y la aplicación está utilizando la interfaz DriverManager para conexiones:
  1. Especifique el nombre de servidor y número de puerto del gestor de conexiones principal que desee utilizar en el URL de conexión.
  2. Utilice los nombres de servidor y números de puerto de los gestores de conexiones alternativos que desee utilizar como valor de las propiedades clientRerouteAlternateServerName y clientRerouteAlternatePortNumber.
- Si el clúster de alta disponibilidad está utilizando más de un gestor de conexiones, y la aplicación está utilizando la interfaz DataSource para conexiones, utilice una de estas técnicas:
  - Defina los nombres de servidor y números de puerto en las propiedades de DataSource:
    1. Utilice el nombre de servidor y número de puerto del gestor de conexiones principal que desee utilizar como valor de las propiedades serverName y portNumber.

2. Utilice los nombres de servidor y números de puerto de los gestores de conexiones alternativos que desee utilizar como valor de las propiedades clientRerouteAlternateServerName y clientRerouteAlternatePortNumber.

– Configure JNDI para la alta disponibilidad utilizando una instancia de DB2ClientRerouteServerList para identificar el gestor de conexiones principal y los gestores de conexiones alternativos.

1. Cree una instancia de DB2ClientRerouteServerList.

   DB2ClientRerouteServerList es un bean de Java serializable con las propiedades siguientes:

Nombre de propiedad	Tipo de datos
com.ibm.db2.jcc.DB2ClientRerouteServerList.alternateServerName	String[]
com.ibm.db2.jcc.DB2ClientRerouteServerList.alternatePortNumber	int[]
com.ibm.db2.jcc.DB2ClientRerouteServerList.primaryServerName	String[]
com.ibm.db2.jcc.DB2ClientRerouteServerList.primaryPortNumber	int[]

   Para cada propiedad se definen los métodos getXXX y setXXX.

2. Utilice el nombre de servidor y número de puerto del gestor de conexiones principal que desee utilizar como valor de las propiedades com.ibm.db2.jcc.DB2ClientRerouteServerList.primaryServerName y com.ibm.db2.jcc.DB2ClientRerouteServerList.primaryPortNumber.
3. Utilice los nombres de servidor y números de puerto de los gestores de conexiones alternativos que desee utilizar como valor de las propiedades com.ibm.db2.jcc.DB2ClientRerouteServerList.alternateServerName y com.ibm.db2.jcc.DB2ClientRerouteServerList.alternatePortNumber.
4. Para hacer que DB2ClientRerouteServerList sea persistente:
   a. Vincule la instancia de DB2ClientRerouteServerList con el registro de JNDI.
   b. Asigne el nombre de JNDI del objeto DB2ClientRerouteServerList a la propiedad clientRerouteServerListJNDIName de IBM Data Server Driver para JDBC y SQLJ.
   c. Asigne el nombre del contexto JNDI que se utiliza para la vinculación y la consulta de la instancia DB2ClientRerouteServerList en la propiedad clientRerouteServerListJNDIContext.

Cuando una fuente de datos DataSource se configura para utilizar JNDI a fin de almacenar información alternativa sobre el redireccionamiento automático del cliente, las propiedades estándar del servidor y puerto de la fuente de datos DataSource no se utilizan para una petición getConnection. En su lugar, se obtiene la dirección del servidor primario a partir de la información transitoria de clientRerouteServerList. Si el almacén de datos JNDI no está disponible debido a un error de vinculación o búsqueda de JNDI, IBM Data Server Driver para JDBC y SQLJ intenta crear una conexión utilizando las propiedades estándar de servidor y puerto de DataSource. Se producen avisos para indicar que se ha producido un error de vinculación o búsqueda de JNDI.

Después de una migración tras error:

- IBM Data Server Driver para JDBC y SQLJ intenta propagar la información actualizada del servidor al almacén de datos JNDI.

- Los valores primaryServerName y primaryPortNumber que se especifican en DB2ClientRerouteServerList se utilizan para la conexión. Si no se especifica el valor primaryServerName, se utilizará el valor serverName de la instancia de DataSource.

Ejemplos de habilitación del soporte de alta disponibilidad de IBM Informix en aplicaciones Java

La configuración del cliente de Java para el soporte de alta disponibilidad de IBM Informix incluye la configuración de varias propiedades de IBM Data Server Driver para JDBC y SQLJ.

El ejemplo siguiente muestra la configuración de aplicaciones cliente de Java para el soporte de alta disponibilidad de IBM Informix.

Antes de configurar el cliente, debe configurar uno o más clústeres de alta disponibilidad que estén controlados por gestores de conexiones.

Siga estos pasos para configurar el cliente:

1. Compruebe que el controlador IBM Data Server Driver para JDBC y SQLJ esté al nivel correcto para utilizar el equilibrado de la carga de trabajo. Para ello siga estos pasos:
   a. Emita el mandato siguiente en una ventana de línea de mandatos:
      ```
      java com.ibm.db2.jcc.DB2Jcc -version
      ```
   b. Busque una línea como la siguiente en los datos de salida y compruebe que *nnn* sea 3.52 o posterior.
   c.
      ```
      [jcc] Driver: IBM Data Server Driver for JDBC and SQLJ Architecture nnn xxx
      ```
2. Establezca las propiedades de IBM Data Server Driver para JDBC y SQLJ para habilitar el concentrador de conexiones o el equilibrado de carga:
   a. Establezca estas propiedades de Connection o DataSource:
      - enableSysplexWLB
      - maxTransportObjects
   b. Establezca la propiedad de configuración global db2.jcc.maxRefreshInterval en un archivo DB2JccConfiguration.properties para establecer el intervalo máximo de renovación para todas las instancias de DataSource o Connection que se crean en el controlador.

   Empiece con una configuración similar a la siguiente:

Tabla 43. Ejemplo de valores de propiedad para el equilibrado de carga de DB2 Database para Linux, UNIX y Windows

Propiedad	Valor
enableSysplexWLB	true
maxTransportObjects	80
db2.jcc.maxRefreshInterval	10

   Los valores que se especifican no se dan como recomendación. Los valores deben determinarse basándose en factores como el número de conexiones físicas disponibles. El número de objetos de transporte debe ser igual o mayor que el número de objetos de conexión.
3. Establezca las propiedades de configuración de IBM Data Server Driver para JDBC y SQLJ para refinar el equilibrado de la carga de trabajo de todas las instancias de DataSource o Connection que se han creado de acuerdo con el

controlador. Establezca las propiedades de configuración en un archivo DB2JccConfiguration.properties siguiendo estos pasos:

a. Cree un archivo DB2JccConfiguration.properties o edite el archivo DB2JccConfiguration.properties existente.
b. Establezca la propiedad de configuración siguiente:
   - db2.jcc.maxTransportObjects

     Comience con un valor similar al siguiente:

     ```
     db2.jcc.maxTransportObjects=1000
     ```
c. Incluya el directorio que contiene DB2JccConfiguration.properties en la concatenación CLASSPATH.

Funcionamiento del redireccionamiento de cliente automático para conexiones con IBM Informix desde clientes de Java

Cuando está habilitado el soporte del redireccionamiento de clientes de IBM Data Server Driver para JDBC y SQLJ, una aplicación Java que esté conectada con un clúster de alta disponibilidad de IBM Informix puede continuar con la ejecución cuando el servidor primario tiene una anomalía.

El redireccionamiento de cliente automático para una aplicación Java que esté conectada con un servidor IBM Informix funciona del mismo modo cuando se habilita el redireccionamiento de cliente automático:

1. Durante cada conexión a la fuente de datos, IBM Data Server Driver para JDBC y SQLJ obtiene información sobre los servidores primarios e intercambiables.
   - Para la primera conexión con IBM Informix:
     a. La aplicación especifica un servidor y un puerto para la conexión inicial. Esos valores identifican un gestor de conexiones.
     b. IBM Data Server Driver para JDBC y SQLJ utiliza la información procedente del gestor de conexiones para obtener información sobre el servidor primario y los servidores alternativos. IBM Data Server Driver para JDBC y SQLJ carga esos valores en la memoria.
     c. Si falla la conexión inicial con el gestor de conexiones:
        – Si están definidas las propiedades clientRerouteAlternateServerName y clientRerouteAlternatePortNumber, IBM Data Server Driver para JDBC y SQLJ se conecta al gestor de conexiones identificado por clientRerouteAlternateServerName y clientRerouteAlternatePortNumber, y obtiene información sobre los servidores primario y alternativos a partir de ese gestor de conexiones. IBM Data Server Driver para JDBC y SQLJ carga esos valores en la memoria como valores de servidor primario y servidor alternativo.
        – Si las propiedades clientRerouteAlternateServerName y clientRerouteAlternatePortNumber no están definidas, y se configura un almacén de datos JNDI mediante la propiedad clientRerouteServerListJNDIName en DB2BaseDataSource, IBM Data Server Driver para JDBC y SQLJ se conecta al gestor de conexiones identificado por DB2ClientRerouteServerList.alternateServerName y DB2ClientRerouteServerList.alternatePortNumber, y obtiene información sobre los servidores primario y alternativos a partir de ese gestor de conexiones. IBM Data Server Driver para JDBC y SQLJ carga en la memoria la información sobre los servidores primario y alternativos obtenida del gestor de conexiones.
     d. Si las propiedades clientRerouteAlternateServerName y clientRerouteAlternatePortNumber no están definidas, y JNDI no está

configurado, IBM Data Server Driver para JDBC y SQLJ consulta tablas DNS para obtener información sobre servidores y puertos del gestor de conexiones. Si existe información de DNS, IBM Data Server Driver para JDBC y SQLJ se conecta al gestor de conexiones, obtiene información sobre servidores primarios y alternativos, y carga esos valores en la memoria.

e. Si no está disponible ninguna información del servidor primario o alternativo, no se puede establecer una conexión e IBM Data Server Driver para JDBC y SQLJ emite una excepción.

- Para conexiones subsiguientes, IBM Data Server Driver para JDBC y SQLJ obtiene valores de los servidores primarios e intercambiables desde la memoria del controlador.

2. IBM Data Server Driver para JDBC y SQLJ intenta conectarse a la fuente de datos utilizando el nombre y el número de puerto del servidor primario.

   Si la conexión se realiza a través de la interfaz DriverManager, IBM Data Server Driver para JDBC y SQLJ crea un objeto DataSource interno para el proceso del redireccionamiento automático de clientes.

3. Si falla la conexión con el servidor primario:
   a. Si se trata de la primera conexión, IBM Data Server Driver para JDBC y SQLJ intenta volverse a conectar con el servidor primario original.
   b. Si no se trata de la primera conexión, IBM Data Server Driver para JDBC y SQLJ intenta volver a conectarse al nuevo servidor primario, cuyo nombre de servidor y número de puerto son facilitados por el servidor.
   c. Si falla la reconexión con el servidor primario, IBM Data Server Driver para JDBC y SQLJ intenta conectar con los servidores alternativos.

      Si esto no es la primera conexión, se utiliza la lista más reciente de servidores alternativos para buscar el servidor alternativo siguiente.

   La conexión con un servidor alternativo se denomina *migración tras error*.

   IBM Data Server Driver para JDBC y SQLJ utiliza las propiedades maxRetriesForClientReroute y retryIntervalForClientReroute para determinar cuántas veces se debe reintentar la conexión y cuánto tiempo se debe esperar entre los reintentos. El intento de conectar con el servidor primario y los servidores alternativos se contabiliza como un solo reintento.

4. Si la conexión no se establece, las propiedades maxRetriesForClientReroute y retryIntervalForClientReroute no están definidas, y los valores originales de serverName y portNumber definidos en DataSource son diferentes de los valores de serverName y portNumber utilizados para la conexión original, reintente la conexión con los valores de serverName y portNumber definidos en DataSource.
5. Si la migración tras error es satisfactoria durante la conexión inicial, el controlador genera un SQLWarning. Si se produce una migración tras error satisfactoria después de la conexión inicial:
   - Si la migración tras error sin fisuras está habilitada, el controlador reintenta la transacción en el nuevo servidor, sin avisar a la aplicación.

     Se deben satisfacer las condiciones siguientes para que se produzca una migración tras error sin fisuras:

     – La propiedad enableSeamlessFailover está establecida en DB2BaseDataSource.YES (1).

       Si el equilibrado de carga de trabajo de Sysplex está vigente (el valor de enableSysplexWLB es `true`), se intenta la migración tras error sin fisuras, independientemente del valor de enableSeamlessFailover.

- La conexión no está en la transacción. Es decir, la anomalía se produce cuando la primera sentencia de SQL se ejecuta en la transacción.
- No existen tablas temporales globales activas en el servidor.
- No hay cursores retenidos abiertos.

- Si la migración tras error sin fisuras no está vigente, el controlador emite una SQLException a la aplicación con el código de error -4498, lo que indica a la aplicación que la conexión se ha restablecido automáticamente y que la transacción se ha retrotraído de forma implícita. A continuación, la aplicación puede intentar su transacción sin realizar una retroacción en primer lugar.

  Un código de razón devuelto con el código de error -4498 indica si los registros especiales del servidor de bases de datos que se han modificado durante la conexión original se restablecerán en la conexión de migración tras error.

Puede determinar si se ha utilizado la información de servidor alternativo al establecer la conexión inicial llamando al método DB2Connection.alternateWasUsedOnConnect.

6. Tras la migración tras error, la memoria del controlador se actualiza con la información del nuevo servidor primario y del nuevo servidor intercambiable a partir del nuevo servidor primario.

Ejemplos

Ejemplo: redireccionamiento de cliente automático a un servidor IBM Informix si todavía no se han definido maxRetriesForClientReroute ni retryIntervalForClientReroute: Suponga que se han definido las propiedades siguientes para una conexión con una base de datos:

Propiedad	Valor
enableClientAffinitiesList	DB2BaseDataSource.NO (2)
serverName	host1
portNumber	port1
clientRerouteAlternateServerName	host2
clientRerouteAlternatePortNumber	port2

Los pasos siguientes demuestran una situación de redireccionamiento automático de cliente para una conexión con IBM Informix:

1. IBM Data Server Driver para JDBC y SQLJ intenta conectar con el gestor de conexiones identificado por host1:port1.
2. La conexión con host1:port1 falla, por lo que el controlador intenta conectar con el gestor de conexiones identificado por host2:port2.
3. La conexión con host2:port2 se realiza satisfactoriamente.
4. El controlador recupera información sobre el servidor alternativo recibida del servidor host2:port2, y actualiza su memoria con esa información.

   Supongamos que el controlador recibe una lista de servidores en la que aparecen host2:port2, host2a:port2a. host2:port2 se almacena como servidor primario nuevo, y host2a:port2a se almacena como servidor alternativo nuevo. Si se detecta otro error de comunicación en esta misma conexión, o en otra conexión creada a partir de la misma DataSource, el controlador intenta conectar con host2:port2 en calidad de nuevo servidor primario. Si esa conexión falla, el controlador intenta conectar con el nuevo servidor alternativo host2a:port2a.

5. El controlador se conecta a host1a:port1a.
6. Se produce un error durante la conexión con host1a:port1a.
7. El controlador intenta conectar con host2a:port2a.
8. La conexión con host2a:port2a es satisfactoria.
9. El controlador recupera información sobre el servidor alternativo recibida del servidor host2a:port2a, y actualiza su memoria con esa información.

Ejemplo: redireccionamiento de cliente automático hacia un servidor IBM Informix cuando las propiedades maxRetriesForClientReroute y retryIntervalForClientReroute están definidas para varios reintentos: Suponga que están definidas las propiedades siguientes para una conexión con una base de datos:

Propiedad	Valor
enableClientAffinitiesList	DB2BaseDataSource.NO (2)
serverName	host1
portNumber	port1
clientRerouteAlternateServerName	host2
clientRerouteAlternatePortNumber	port2
maxRetriesForClientReroute	3
retryIntervalForClientReroute	2

Los pasos siguientes demuestran una situación de redireccionamiento automático de cliente para una conexión con IBM Informix:

1. IBM Data Server Driver para JDBC y SQLJ intenta conectar con el gestor de conexiones identificado por host1:port1.
2. La conexión con host1:port1 falla, por lo que el controlador intenta conectar con el gestor de conexiones identificado por host2:port2.
3. La conexión con host2:port2 se realiza satisfactoriamente.
4. El controlador recupera información sobre el servidor alternativo a partir del gestor de conexiones identificado por host2:port2, y actualiza su memoria con esa información. Suponga que el gestor de conexiones identifica host1a:port1a como nuevo servidor primario, y host2a:port2a como nuevo servidor alternativo.
5. El controlador intenta conectar con host1a:port1a.
6. La conexión con host1a:port1a falla.
7. El controlador intenta conectar con host2a:port2a.
8. La conexión con host2a:port2a falla.
9. El controlador espera dos segundos.
10. El controlador intenta conectar con host1a:port1a.
11. La conexión con host1a:port1a falla.
12. El controlador intenta conectar con host2a:port2a.
13. La conexión con host2a:port2a falla.
14. El controlador espera dos segundos.
15. El controlador intenta conectar con host1a:port1a.
16. La conexión con host1a:port1a falla.
17. El controlador intenta conectar con host2a:port2a.
18. La conexión con host2a:port2a falla.

19. El controlador espera dos segundos.
20. El controlador emite una excepción de SQL (SQLException) que incluye el código de error -4499.

Funcionamiento del equilibrado de la carga de trabajo para conexiones con IBM Informix de clientes de Java

El equilibrado de la carga de trabajo, denominado también equilibrado de la carga de trabajo en el nivel de transacción, para conexiones con IBM Informix contribuye a la alta disponibilidad al equilibrar el trabajo entre los servidores en un clúster de alta disponibilidad en el comienzo de una transacción.

En la visión general siguiente se describen los pasos que tienen lugar cuando un cliente se conecta con un gestor de conexiones de IBM Informix y se habilita el equilibrado de la carga de trabajo:

1. Cuando el cliente establece en primer lugar una conexión mediante la dirección IP del gestor de conexiones, éste devuelve la lista de servidores y los detalles de la conexión (dirección IP, puerto y peso) para los servidores en el clúster.

   El cliente guarda la lista de servidores en la memoria caché. La vida útil por omisión de la lista de servidores en la memoria caché es de 30 segundos.
2. Al inicio de una transacción nueva, el cliente lee la lista de servidores almacenada en la memoria caché para identificar a un servidor que tenga capacidad no intervenida y busca en la agrupación de transporte un transporte desocupado que esté vinculado al servidor infrautilizado. (Un transporte desocupado es un transporte que no tiene ningún objeto de conexión asociado).
   - Si un transporte desocupado está disponible, el cliente asocia el objeto de conexión con el transporte.
   - Si, después de un tiempo de espera excedido configurable por el usuario, no hay ningún transporte desocupado disponible en la agrupación de transporte y no puede asignarse ningún transporte nuevo puesto que la agrupación de transporte ha alcanzado su límite, se devuelve un error a la aplicación.
3. Cuando la transacción se ejecuta, accede al servidor que está vinculado al transporte.

   Cuando la primera sentencia de SQL en una transacción se ejecuta, si IBM Data Server Driver para JDBC y SQLJ recibe un error de comunicación porque el servidor de datos desactiva la conexión o se sobrepasa el valor blockingReadConnectionTimeout, el controlador vuelve a intentar la sentencia de SQL 10 veces antes de notificar un error. En cada reintento, el controlador cierra el transporte existente, obtiene un nuevo transporte y, a continuación, ejecuta la transacción. Durante estos reintentos, si las propiedades maxRetriesForClientReroute y retryIntervalForClientReroute están establecidas, sus valores sólo se aplican al proceso de obtener un nuevo transporte durante cada reintento.
4. Cuando la transacción finaliza, el cliente verifica con el servidor que la reutilización del transporte aún está permitida para el objeto de conexión.
5. Si la reutilización del transporte aún está permitida, el servidor devolverá una lista de sentencias SET para registros especiales que se apliquen al entorno de ejecución para el objeto de conexión.

   El cliente almacena en la antememoria estas sentencias, que reproduce con el fin de reconstruir el entorno de ejecución cuando el objeto de conexión está asociado con un transporte nuevo.
6. El objeto de la conexión se desasocia entonces del transporte, si el cliente determina que necesita hacerlo.

7. La copia de cliente de la lista de servidores se renueva cuando se establece una nueva conexión, cada 30 segundos o según un intervalo configurado por el usuario.
8. Cuando el equilibrado de la carga de trabajo es obligatorio para una transacción nueva, el cliente utiliza el proceso descrito anteriormente para asociar el objeto de conexión con un transporte.

Requisitos de programación de aplicaciones para obtener una alta disponibilidad para conexiones de clientes de Java con servidores IBM Informix

La migración tras error para el redireccionamiento de cliente automático puede ser con o sin fisuras. Si la migración tras error para las conexiones con IBM Informix no se produce sin fisuras, debe agregar el código para tener en cuenta los errores que se han devuelto cuando se produce la migración tras error.

Si una migración tras error no se produce sin fisuras y se vuelve a establecer una conexión con el servidor, SQLCODE -4498 (para clientes de Java) o SQL30108N (para clientes que no son de Java) se devuelve a la aplicación. Todo el trabajo que se ha producido en la transacción actual se ha retrotraído. En la aplicación, necesita:

- Compruebe el código de razón que se devuelve con el error. Determine si se transfieren los valores de registros especiales en el miembro de compartimiento de datos que falla al nuevo miembro de compartimiento de datos (migración tras error). Restaure cualquier valor de registro especial que no sea actual.
- Ejecute todas las operaciones de SQL que han tenido lugar durante la transacción anterior.

Se deben cumplir las condiciones siguientes para que se produzca una migración tras error sin fisuras durante las conexiones a las bases de datos IBM Informix:

- El lenguaje de programación de la aplicación es Java, CLI o .NET.
- La conexión no está en la transacción. Es decir, la anomalía se produce cuando la primera sentencia de SQL se ejecuta en la transacción.
- El servidor de datos debe permitir la reutilización del transporte al final de la transacción anterior.
- Todos los datos de sesiones globales se han cerrado o descartado.
- No hay cursores retenidos abiertos.
- Si la aplicación utiliza CLI, ésta no podrá llevar a cabo acciones que requieran al controlador que mantenga un historial de las API a las que se ha llamado anteriormente, con el fin de reproducir la sentencia de SQL. Ejemplos de dichas acciones son la especificación de datos durante la ejecución, ejecutar sentencias de SQL compuesto o utilizar entradas de matrices.
- La aplicación no es un procedimiento almacenado.
- Autocommit no se ha habilitado. La migración tras error sin fisuras puede producirse cuando se habilita Autocommit. Sin embargo, la situación siguiente puede producir problemas: suponga que la tarea de SQL se ejecuta satisfactoriamente y se confirma en el servidor de datos, pero la conexión o el servidor queda inactivo antes de que el reconocimiento de la operación de confirmación se vuelva a enviar al cliente. Cuando el cliente restablece la conexión, vuelve a reproducir la sentencia de SQL confirmada anteriormente. El resultado es que la sentencia de SQL se ejecuta dos veces. Para evitar esta situación, desactive la confirmación automática cuando habilite la migración tras error sin fisuras.

Asimismo, el redireccionamiento de cliente automático sin fisuras podría no resultar satisfactorio si la aplicación tiene habilitada la confirmación automática. Con la confirmación automática habilitada, una sentencia puede ejecutarse y confirmarse varias veces.

Afinidades de cliente para conexiones con IBM Informix de clientes de Java

Afinidades de cliente es un método sólo para clientes para proporcionar funciones de redireccionamiento de cliente automático.

Hay afinidades de cliente disponibles para aplicaciones que utilicen CLI, .NET o Java (IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4). El redireccionamiento está controlado por el controlador.

Las afinidades de cliente están pensadas para situaciones donde debe conectarse con un servidor primario determinado. Si se produce una parada durante la conexión con el servidor primario, debe imponer un orden específico para la migración tras error a servidores alternativos. Debe utilizar afinidades de cliente para el redireccionamiento de cliente automático, sólo si el redireccionamiento de cliente automático que utiliza las funciones de migración tras error del servidor no funciona en su entorno.

Como parte de la configuración de afinidades de cliente, especifique una lista de servidores alternativos, así como el orden en que se probarán las conexiones con los servidores alternativos. Cuando las afinidades de cliente estén en uso, las conexiones se establecen de acuerdo con la lista de servidores alternativos, en lugar del nombre de host y el número de puerto especificados por la aplicación. Por ejemplo, si una aplicación especifica que se ha establecido una conexión con server1, pero el proceso de configuración especifica que los servidores deben probarse en el orden (server2, server3, server1), la conexión inicial se establece con server2 en lugar de con server1.

La migración tras error con afinidades de cliente será sin fisuras, si se dan las condiciones siguientes:

- La conexión no está en la transacción. Es decir, la anomalía se produce cuando la primera sentencia de SQL se ejecuta en la transacción.
- No existen tablas temporales globales activas en el servidor.
- No hay cursores retenidos abiertos.

Si utiliza afinidades de cliente, puede especificar que si el servidor primario vuelve a estar operativo después de una parada, las conexiones vuelven de un servidor alternativo al servidor primario en un límite de transacción. Esta actividad recibe el nombre de *recuperación*.

Configuración de afinidades de cliente de Java para conexiones de IBM Informix

Para habilitar el soporte de afinidades de cliente en aplicaciones Java, defina las propiedades para indicar que desea usar las afinidades de cliente y para especificar los servidores primarios y alternativos.

La tabla siguiente describe los valores de las propiedades para habilitar afinidades de cliente para aplicaciones Java.

Tabla 44. Valores de propiedades para habilitar afinidades de cliente para aplicaciones Java

Valor de IBM Data Server Driver para JDBC y SQLJ	Valor
enableClientAffinitiesList	DB2BaseDataSource.YES (1)
clientRerouteAlternateServerName	Una lista separada por comas de los servidores primarios y alternativos
clientRerouteAlternatePortNumber	Una lista separada por comas de los números de puertos de los servidores primarios y alternativos
enableSeamlessFailover	DB2BaseDataSource.YES (1) para migración tras error sin fisuras; DB2BaseDataSource.NO (2) o enableSeamlessFailover sin especificar para ninguna migración tras error sin fisuras
maxRetriesForClientReroute	Número de veces que se volverá a intentar la conexión con cada servidor, incluido el servidor primario, después de que falle una conexión con el servidor primario. El valor por omisión es 3.
retryIntervalForClientReroute	Número de segundos que se debe esperar entre reintentos. El valor por omisión es sin espera.
affinityFailbackInterval	Número de segundos que se debe esperar después de que el primer límite de transacción se transfiera al servidor primario. Defina este valor si desea realizar la transferencia al servidor primario.

Ejemplo de habilitación de afinidades en clientes de Java para conexiones de IBM Informix

Para utilizar las afinidades de cliente en el redireccionamiento de cliente automático en aplicaciones Java, debe definir propiedades para indicar que desea utilizar afinidades de cliente, así como para identificar los servidores alternativos y primarios.

El ejemplo siguiente muestra cómo habilitar afinidades de cliente para una migración tras error sin transferencia.

Suponga que establece las propiedades siguientes para una conexión con una base de datos:

Propiedad	Valor
enableClientAffinitiesList	DB2BaseDataSource.YES (1)
clientRerouteAlternateServername	host1,host2,host3
clientRerouteAlternatePortNumber	port1,port2,port3
maxRetriesForClientReroute	3
retryIntervalForClientReroute	2

Suponga que se produce un error de comunicación durante una conexión con el servidor identificado por host1:port1. Los pasos siguientes muestran el redireccionamiento de cliente automático con afinidades de cliente.

1. El controlador intenta conectar con host1:port1.

2. La conexión con host1:port1 falla.
3. El controlador espera dos segundos.
4. El controlador intenta conectar con host1:port1.
5. La conexión con host1:port1 falla.
6. El controlador espera dos segundos.
7. El controlador intenta conectar con host1:port1.
8. La conexión con host1:port1 falla.
9. El controlador espera dos segundos.
10. El controlador intenta conectar con host2:port2.
11. La conexión con host2:port2 falla.
12. El controlador espera dos segundos.
13. El controlador intenta conectar con host2:port2.
14. La conexión con host2:port2 falla.
15. El controlador espera dos segundos.
16. El controlador intenta conectar con host2:port2.
17. La conexión con host2:port2 falla.
18. El controlador espera dos segundos.
19. El controlador intenta conectar con host3:port3.
20. La conexión con host3:port3 falla.
21. El controlador espera dos segundos.
22. El controlador intenta conectar con host3:port3.
23. La conexión con host3:port3 falla.
24. El controlador espera dos segundos.
25. El controlador intenta conectar con host3:port3.
26. La conexión con host3:port3 falla.
27. El controlador espera dos segundos.
28. El controlador emite una excepción de SQL (SQLException) que incluye el código de error -4499.

El ejemplo siguiente muestra cómo habilitar afinidades de cliente para una migración tras error con transferencia.

Suponga que establece las propiedades siguientes para una conexión con una base de datos:

Propiedad	Valor
enableClientAffinitiesList	DB2BaseDataSource.YES (1)
clientRerouteAlternateServername	host1,host2,host3
clientRerouteAlternatePortNumber	port1,port2,port3
maxRetriesForClientReroute	3
retryIntervalForClientReroute	2
affinityFailbackInterval	300

Suponga que el administrador de la base de datos retira el servidor identificado por host1:port1 a mantenimiento, después de que se establezca una conexión con

host1:port1. Los pasos siguientes muestran la migración tras error a un servidor alternativo y la retrotracción al servidor primario después de que se haya completado el mantenimiento.

1. El controlador se conecta satisfactoriamente con host1:port1 en nombre de una aplicación.
2. El administrador de la base de datos desactiva host1:port1.
3. La aplicación intenta realizar tareas en la conexión.
4. El controlador transfiere satisfactoriamente el control a host2:port2.
5. Cuando han transcurrido 200 segundos, la tarea se confirma.
6. Cuando han transcurrido 300 segundos, ha transcurrido el intervalo de recuperación. El controlador comprueba si el servidor primario está activo. No está activo, así que no se produce la recuperación.
7. Cuando han transcurrido 350 segundos, host1:port1 vuelve a quedar en línea.
8. La aplicación continúa realizando tareas en host2:port2, porque no ha transcurrido el intervalo de recuperación más reciente.
9. Cuando han transcurrido 600 segundos, ha vuelto a transcurrir el intervalo de recuperación. El controlador comprueba si el servidor primario está activo. Ahora está activo.
10. Cuando han transcurrido 650 segundos, la tarea se confirma.
11. Cuando han transcurrido 651 segundos, la aplicación intenta iniciar una nueva transacción en host2:port2. Se produce la recuperación en el puerto host1:port1; por lo tanto, la nueva transacción se inicia en host1:port1.

Soporte de conexión directa de cliente de Java a fin de obtener alta disponibilidad para conexiones con servidores DB2 para z/OS

Las funciones de equilibrado de la carga de trabajo de Sysplex en los servidores DB2 para z/OS proporcionan una alta disponibilidad a aplicaciones cliente que conectan directamente a un grupo de compartimiento de datos. Las funciones de equilibrado de la carga de trabajo de Sysplex ofrecen el equilibrado de la carga de trabajo y la capacidad de redireccionamiento automático del cliente. Este soporte está disponible para aplicaciones que usan clientes de Java (JDBC, SQLJ o pureQuery) que utilizan IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4, o bien clientes que no son de Java (ODBC, CLI, .NET, OLE DB, PHP, Ruby o SQL incorporado). El equilibrado de la carga de trabajo es transparente para las aplicaciones.

Un Sysplex es un conjunto de sistemas z/OS que se comunican y cooperan entre sí a través de determinados componentes de hardware multisistema y servicios de software para procesar cargas de trabajo de clientes. Los subsistemas de DB2 para z/OS en los sistemas z/OS de un Sysplex se pueden configurar para que formen un grupo de compartimiento de datos. Gracias al compartimiento de datos, las aplicaciones que se ejecuten en más de un subsistema de DB2 para z/OS puede leer y escribir en el mismo conjunto de datos simultáneamente. Uno o más recursos de acoplamiento proporcionan almacenamiento de antememoria de alta velocidad y procesos de bloqueo para el grupo de compartimiento de datos. El Sysplex, junto con el gestor de carga de trabajo (WLM), las direcciones IP virtuales dinámicas (DVIPA) y Sysplex Distributor, permite a un cliente acceder a una base de datos DB2 para z/OS a través de TCP/IP con flexibilidad de red, y distribuir transacciones para una aplicación de manera equilibrada entre miembros del grupo de compartimiento de datos.

Para estas características es indispensable una lista de servidores que el grupo de compartimiento de datos devuelve en los límites de las conexiones y de forma opcional en los límites de las transacciones. Esta lista contiene la dirección IP y el peso de WLM para cada miembro del grupo de compartimiento de datos. Mediante esta información, un cliente puede distribuir transacciones de un modo equilibrado, o bien identificar qué miembro se debe usar cuando haya una anomalía en la comunicación.

La lista de servidores se devuelve en la primera conexión satisfactoria con el servidor de datos de DB2 para z/OS. Después de que el cliente haya recibido la lista de servidores, el cliente accede directamente al miembro del grupo de compartimiento de datos según la información de la lista de servidores.

DB2 para z/OS ofrece varios métodos para que los clientes accedan a un grupo de compartimiento de datos. El método de acceso que está configurado para la comunicación con el grupo de compartimiento de datos determina si es posible el equilibrado de la carga de trabajo de Sysplex. La tabla siguiente enumera los métodos de acceso e indica si es posible el equilibrado de la carga de trabajo de Sysplex.

Tabla 45. Equilibrado de la carga de trabajo de Sysplex y métodos de acceso de compartimiento de datos

Método de acceso de compartimiento de datos[1]	**Descripción**	**¿Equilibrado de la carga de trabajo de Sysplex posible?**
Acceso de grupos	Un peticionario utiliza la dirección IP de grupo DB2 para realizar una conexión inicial con la ubicación de DB2 para z/OS. Una conexión al grupo de compartimiento de datos que utiliza el puerto SQL y la dirección IP de grupo siempre es satisfactoria si se inicia un miembro como mínimo. La lista de servidores que devuelve el grupo de compartimiento de datos contiene: • Una lista de miembros que se encuentran activos y pueden trabajar • El peso de WLM de cada miembro La dirección IP de grupo está configurada mediante el distribuidor de Sysplex de z/OS. Para los clientes que se hallan fuera del Sysplex, el distribuidor de Sysplex proporciona una única dirección IP que representa una ubicación de DB2. Además de ofrecer tolerancia de errores, el distribuidor de Sysplex puede configurarse para proporcionar equilibrado de carga de conexión.	Sí

Tabla 45. Equilibrado de la carga de trabajo de Sysplex y métodos de acceso de compartimiento de datos (continuación)

Método de acceso de compartimiento de datos[1]	Descripción	¿Equilibrado de la carga de trabajo de Sysplex posible?
Acceso específico de miembros	Un peticionario utiliza un alias de ubicación para realizar una primera conexión con uno de los miembros representado por ese alias. Una conexión al grupo de compartimiento de datos que utiliza la dirección IP de grupo y el puerto SQL alias siempre es satisfactoria si se inicia un miembro como mínimo. La lista de servidores que devuelve el grupo de compartimiento de datos contiene: • Una lista de miembros que se encuentran activos, pueden trabajar y se han configurado como un alias • El peso de WLM de cada miembro El peticionario utiliza esta información para conectarse al miembro o miembros con la mayor capacidad que también estén asociados con el alias de ubicación. El acceso específico de miembros se utiliza cuando los peticionarios necesitan beneficiarse del equilibrado de la carga de trabajo de Sysplex entre un subconjunto de miembros de un grupo de compartimiento de datos.	Sí
Acceso de miembro único	El acceso de miembro único se usa cuando los peticionarios necesitan acceder solamente a un miembro de un grupo de compartimiento de datos. Para el acceso de miembro único, la conexión utiliza la dirección IP específica para miembros.	No

Nota:

1. Para obtener más información sobre los métodos de acceso de compartimiento de datos, consulte http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db29.doc.dshare/db2z_tcpipaccessmethods.htm.

El equilibrado de la carga de trabajo de Sysplex incluye redireccionamiento de cliente automático: el soporte del redireccionamiento de cliente automático permite a un cliente recuperarse de una anomalía intentando reconectarse a la base de datos a través de cualquier miembro disponible de un Sysplex. La reconexión con otro miembro recibe el nombre de *migración tras error*.

Equilibrado de la carga de trabajo de Sysplex durante la migración de un grupo de compartimiento de datos a DB2 9.1 para z/OS o DB2 10 para z/OS: En general, si se utiliza IBM Data Server Driver para JDBC y SQLJ Versión 3.61 o 4.11, la migración de un grupo de compartimiento de datos de DB2 para z/OS Versión 8 o Versión 9.1 a la Versión 10, o de DB2 para z/OS Versión 8 a la Versión 9.1, no fuerza la interrupción de las aplicaciones Java que se conectan con el grupo de compartimiento de datos que utiliza IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4. No tendrá que reiniciar todos los miembros del grupo de compartimiento de datos o las JVM para mantener equilibradas las conexiones. Además, si utiliza IBM Data Server Driver para JDBC y SQLJ Versión 3.62 o 4.12 o

versiones posteriores en cualquier modalidad durante la migración desde la modalidad de función nueva DB2 para z/OS Versión 8 o Versión 9.1 a la modalidad de función nueva Versión 10, o desde la modalidad de función nueva de DB2 para z/OS Versión 8 a la modalidad de función nueva de la Versión 9.1, las aplicaciones nuevas que utilizan funciones que requieren un nivel de DRDA mayor pueden coexistir con aplicaciones antiguas que utilizan un nivel de DRDA menor, si emplean la misma DataSource. Esta coexistencia incluye la reversión de una modalidad a la anterior, por ejemplo la reversión de la ENFM9 Versión 10 a CM9*. En lo que respecta a la coexistencia de los niveles de DRDA, hay que tener instalado el APAR PM24292 en los servidores de datos DB2 para z/OS Versión 9.1 y DB2 para z/OS Versión 10.

Equilibrado de la carga de trabajo de Sysplex durante la migración de un grupo de compartimiento de datos a DB2 9.1 para z/OS: cuando migre un grupo de compartimiento de datos a DB2 9.1 para z/OS en la modalidad de función nueva, deberá seguir estos pasos:

1. Reinicie todos los miembros del grupo de datos.
2. Reinicie las JVM en las que se ejecuten aplicaciones que se conecten al grupo de compartimiento de datos mediante IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4.

La detención y el inicio de todos los miembros evitan que las aplicaciones que utilizan el equilibrado de la carga de trabajo de Sysplex tengan conexiones no equilibradas.

Para las aplicaciones cliente de Java, CLI o .NET, la migración tras error para el redireccionamiento de cliente automático puede ser *sin fisuras* o *con fisuras*. La migración tras error sin fisuras significa que cuando la aplicación se reconecta satisfactoriamente con un servidor alternativo, éste no devuelve un error a la aplicación.

Soporte de conexión directa de cliente para obtener alta disponibilidad con un DB2 Connect Server: el soporte de conexión directa para obtener alta disponibilidad exige una licencia de DB2 Connect, pero no necesita un DB2 Connect Server. El cliente se conecta directamente con DB2 para z/OS. Si utiliza un DB2 Connect Server, pero configura el entorno para obtener alta disponibilidad de cliente, no podrá beneficiarse de algunas de las características que una conexión directa con DB2 para z/OS proporciona, como el equilibrado de carga de trabajo de nivel de transacción o la prestación de redireccionamiento automático de clientes que proporciona el Sysplex.

No utilice afinidades de cliente: no se deben usar afinidades de cliente como solución de alta disponibilidad para las conexiones directas con DB2 para z/OS. Las afinidades de cliente no son aplicables a un entorno de compartimiento de datos de DB2 para z/OS, porque todos los miembros de un grupo de compartimiento de datos puede acceder a los datos de forma simultánea. Una desventaja importante de las afinidades de clientes en un entorno de compartimiento de datos es que si la migración tras error se produce porque falla un miembro del grupo de compartimiento de datos, éste puede haber retenido bloqueos que pueden afectar gravemente a las transacciones en el miembro en el que tiene lugar la migración tras error.

Configuración del equilibrado de la carga de trabajo de Sysplex y del redireccionamiento de cliente automático para clientes de Java

Para configurar una aplicación cliente de IBM Data Server Driver para JDBC y SQLJ que se conecte directamente con DB2 para z/OS para utilizar el equilibrado de la carga de trabajo de Sysplex y del redireccionamiento de cliente automático, debe utilizar IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4. También debe conectarse con una dirección que represente el grupo de compartimiento de datos (para acceso en grupo) o un subconjunto del grupo de compartimiento de datos (para acceso específico de miembros) y establecer las propiedades que habilitan el equilibrado de la carga de trabajo y el número máximo de conexiones.

Siempre debe configurar conjuntamente el equilibrado de la carga de trabajo de Sysplex y del redireccionamiento de cliente automático. Cuando configura un cliente para que utilice el equilibrado de la carga de trabajo de Sysplex, el redireccionamiento de cliente automático también se habilita. Por lo tanto, sólo debe cambiar los valores de las propiedades que están relacionadas con el redireccionamiento de cliente automático para ajustar con precisión el funcionamiento del redireccionamiento de cliente automático.

La tabla siguiente describe los valores básicos de las propiedades para aplicaciones Java.

Tabla 46. Valores básicos para habilitar el soporte de alta disponibilidad de Sysplex en aplicaciones Java

Método de acceso de compartimiento de datos	**Valor de IBM Data Server Driver para JDBC y SQLJ**	**Valor**
Acceso de grupos	Propiedad enableSysplexWLB	true
	Dirección de conexión:	
	servidor	La dirección IP de grupo o nombre de dominio del grupo de compartimiento de datos
	puerto	Número de puerto SQL para la ubicación de DB2
	basedatos	El nombre de la ubicación de DB2 que se define durante la instalación.
Acceso específico de miembros	Propiedad enableSysplexWLB	true
	Dirección de conexión:	
	servidor	La dirección IP de grupo o nombre de dominio del grupo de compartimiento de datos
	puerto	El número de puerto para el alias de la ubicación de DB2
	basedatos	El nombre del alias de la ubicación de DB2 que representa un subconjunto de los miembros del grupo de compartimiento de datos

Tabla 46. Valores básicos para habilitar el soporte de alta disponibilidad de Sysplex en aplicaciones Java (continuación)

Método de acceso de compartimiento de datos	Valor de IBM Data Server Driver para JDBC y SQLJ	Valor
Acceso de grupo o acceso de miembro específico	commandTimeout	Especifica el tiempo máximo en segundos que una aplicación que se ejecuta bajo IBM Data Server Driver para JDBC y SQLJ espera a que se complete cualquier tipo de petición al servidor de datos antes de que el controlador emita una SQLException. El tiempo de espera incluye el tiempo para obtener un transporte, realizar la migración tras error si es necesario, enviar la petición y esperar una respuesta. El valor por omisión es 0.
	connectionTimeout	Especifica el tiempo máximo en segundos que IBM Data Server Driver para JDBC y SQLJ espera una respuesta del grupo de compartimiento de datos cuando el controlador intenta establecer una conexión. Si el controlador no recibe una respuesta después de la cantidad de tiempo que se ha especificado mediante connectionTimeout, emite una SQLException con el código de error -4499 de SQL. El valor por omisión es 0. Si connectionTimeout se establece en un valor positivo, ese valor altera temporalmente cualquier otro valor de tiempo de espera que se haya establecido en una conexión, como loginTimeout.
Acceso de grupo o acceso de miembro específico	maxTransportObjects	Especifica el número máximo de conexiones que el peticionario puede realizar al grupo de compartimiento de datos. El valor por omisión es 1000. Para determinar el valor maxTransportObjects, multiplique el número esperado de conexiones activas simultáneas con el grupo de compartimiento de datos de DB2 para z/OS por el número de miembros del grupo de compartimiento de datos.

Hay propiedades adicionales disponibles para el equilibrado de la carga de trabajo de Sysplex y el redireccionamiento de cliente automático. Inicialmente, deberá configurar el equilibrado de la carga de trabajo de Sysplex utilizando únicamente las propiedades básicas. En la mayoría de los casos, no es necesario establecer ninguna de las propiedades adicionales.

Las propiedades Connection o DataSource de IBM Data Server Driver para JDBC y SQLJ siguientes se pueden utilizar para ajustar el equilibrado de la carga de Sysplex y el redireccionamiento de cliente automático:
- blockingReadConnectionTimeout
- enableSeamlessFailover
- loginTimeout
- maxRetriesForClientReroute
- memberConnectTimeout
- retryIntervalForClientReroute

Las propiedades de configuración de IBM Data Server Driver para JDBC y SQLJ siguientes se pueden utilizar para ajustar el equilibrado de la carga de Sysplex y el redireccionamiento de cliente automático:
- db2.jcc.maxRefreshInterval
- db2.jcc.maxTransportObjectIdleTime
- db2.jcc.maxTransportObjectWaitTime
- db2.jcc.minTransportObjects

Ejemplo de habilitación del equilibrado de la carga de trabajo de Sysplex y del redireccionamiento de cliente automático de DB2 para z/OS en aplicaciones Java

La configuración del cliente de Java para el equilibrado de la carga de trabajo de Sysplex y el redireccionamiento de cliente automático incluye la configuración de varias propiedades de IBM Data Server Driver para JDBC y SQLJ.

En los ejemplos siguientes se muestra la configuración de las aplicaciones cliente de Java para el equilibrado de la carga de trabajo de Sysplex y el redireccionamiento de cliente automático para alta disponibilidad.

Antes de configurar el cliente, debe configurar el siguiente software de servidor:
- WLM para z/OS

  Para que el equilibrado de la carga de trabajo funcione eficazmente, es necesario clasificar el trabajo de DB2. La clasificación se aplica a la primera sentencia de SQL no establecida en cada transacción. Entre las áreas en las que se debe clasificar el trabajo se encuentran:
  - ID de autorización
  - Propiedades de información del cliente
  - Nombre del procedimiento almacenado

    El nombre del procedimiento almacenado se utiliza con fines de clasificación sólo si la primera sentencia que emite el cliente en la transacción es una sentencia de SQL CALL.

  Para obtener una lista completa de atributos de clasificación, consulte la información sobre atributos de clasificación en el URL siguiente:

  `http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/-topic/com.ibm.db2z10.doc.perf/src/tpc/db2z_classificationattributes.htm`
- DB2 para z/OS, configurado para el compartimiento de datos

Ejemplo de configuración con WebSphere Application Server

En este ejemplo se presupone que utiliza WebSphere Application Server. 5.1 es la versión mínima de WebSphere Application Server.

Siga estos pasos para configurar el cliente:

1. Compruebe que IBM Data Server Driver para JDBC y SQLJ esté al nivel correcto para utilizar el equilibrado de la carga de trabajo de Sysplex. Para ello siga estos pasos:
   a. Emita el mandato siguiente en una ventana de línea de mandatos

      ```
      java com.ibm.db2.jcc.DB2Jcc -version
      ```

   b. Busque una línea como la siguiente en los datos de salida y compruebe que *nnn* es 3.50 o posterior.

      ```
      [jcc] Driver: IBM Data Server Driver for JDBC and SQLJ Architecture nnn xxx
      ```
2. En la consola administrativa de WebSphere Application Server, establezca la propiedad de fuente de datos de IBM Data Server Driver para JDBC y SQLJ

enableSysplexWLB en `true` para habilitar el equilibrado de la carga de trabajo de Sysplex. Por omisión, la habilitación del equilibrado de la carga de trabajo de Sysplex habilita el redireccionamiento de cliente automático.

Tabla 47. Ejemplo de configuración de las propiedades de fuente de datos para el equilibrado de la carga de trabajo de Sysplex y el redireccionamiento automático de cliente de IBM Data Server Driver para JDBC y SQLJ para DB2 para z/OS

Propiedad	Valor
enableSysplexWLB	true
maxTransportObjects	80[1]

Nota:

1. Establezca maxTransportObjects en el producto resultante de multiplicar el número de conexiones que acceden simultáneamente al grupo de compartimiento de datos y el número de miembros del grupo de compartimiento de datos.

3. En la consola administrativa de WebSphere Application Server, establezca otras propiedades para las que se considere que los valores por omisión no son aceptables. Modifique estas propiedades de conexión de WebSphere Application Server. Cuando enableSysplexWLB se establece en `true`, los valores recomendados son los siguientes:

Propiedad de conexión	Valor recomendado	Descripción
Tiempo de recopilación	0	Especifica el intervalo, en segundos, entre ejecuciones de la hebra de mantenimiento de agrupación. El intervalo de tiempo de recopilación afecta al rendimiento. Puesto que las conexiones no son conexiones físicas, se recomienda inhabilitar el mantenimiento de agrupación estableciendo este valor en 0.
Tiempo de espera excedido	0	Especifica el intervalo de tiempo, en segundos, previo al descarte de una conexión física. El establecimiento del tiempo de espera excedido en 0 permite que las conexiones sigan existiendo de forma indefinida en la agrupación.
Política de depuración	FailingConnectionOnly sin soporte de grupos alternativos; EntirePool con soporte de grupos alternativos.	Especifica cómo han de depurarse las conexiones cuando se detecte una conexión obsoleta o un error de conexión muy grave. Puesto que el equilibrado de la carga de trabajo aísla a WebSphere Application Server de las conexiones obsoletas o los errores de conexión muy graves, el valor recomendado es FailingConnectionOnly. Sin embargo, si se ha habilitado el soporte de grupos alternativos, el valor recomendado es EntirePool. Si se produce la migración tras error a otro grupo, el valor EntirePool fuerza todas las conexiones para que tenga lugar la migración de toda la agrupación al grupo alternativo.

No es necesario cambiar el valor de conexiones máximas. Las conexiones de miembro especifican el número máximo de conexiones físicas que crea en la agrupación. No controlan el número de conexiones físicas para un grupo de compartimiento de datos. Con el equilibrado de la carga de trabajo Sysplex, las conexiones son lógicas, y utilizan transportes para asociar una conexión a un miembro de compartimiento de datos. Las conexiones físicas se gestionan mediante agrupaciones de transporte en el controlador. La propiedad maxTransportObjects controla el número máximo de conexiones para el grupo.

4. Establezca las propiedades de configuración de IBM Data Server Driver para JDBC y SQLJ para refinar el equilibrado de la carga de trabajo para todas las instancias de DataSource o Connection que se han creado bajo el controlador. Establezca las propiedades de configuración en un archivo DB2JccConfiguration.properties siguiendo estos pasos:
   a. Cree un archivo DB2JccConfiguration.properties o edite el archivo DB2JccConfiguration.properties existente.

b. Establezca la propiedad de configuración db2.jcc.maxTransportObjects sólo si se definen varios objetos DataSource que se refieran al mismo grupo de compartimiento de datos, y deba limitarse el número de conexiones en los diferentes objetos DataSource.

   Comience con un valor similar al siguiente:

   ```
   db2.jcc.maxTransportObjects=500
   ```

c. Establezca la propiedad de configuración db2.jcc.maxRefreshInterval. Esta propiedad requiere la versión 3.58 o posterior de IBM Data Server Driver para JDBC y SQLJ.

   Comience con un valor similar al siguiente:

   ```
   db2.jcc.maxRefreshInterval=10
   ```

d. Añada la vía de acceso del directorio para DB2JccConfiguration.properties a la classpath de IBM Data Server Driver para JDBC y SQLJ de WebSphere Application Server.

e. Reinicie WebSphere Application Server.

Ejemplo de configuración para conexiones DriverManager

En este ejemplo se presupone que utiliza la interfaz de DriverManager para establecer una conexión.

Siga estos pasos para configurar el cliente:

1. Verifique que IBM Data Server Driver para JDBC y SQLJ corresponde al nivel correcto para que pueda dar soporte al equilibrado de la carga de trabajo de Sysplex y al redireccionamiento automático de cliente realizando estos pasos:
   a. Emita el mandato siguiente en una ventana de línea de mandatos

      ```
      java com.ibm.db2.jcc.DB2Jcc -version
      ```

   b. Busque una línea como la siguiente en los datos de salida y compruebe que *nnn* es 3.50 o posterior. Se necesita el nivel mínimo de controlador 3.50 para poder utilizar el equilibrado de la carga de trabajo de Sysplex y el redireccionamiento automático de cliente para las conexiones de DriverManager.

   c.

      ```
      [jcc] Driver: IBM Data Server Driver for JDBC and SQLJ Architecture nnn xxx
      ```

2. Configure la propiedad enableSysplexWLB de IBM Data Server Driver para JDBC y SQLJ Connection para habilitar el equilibrado de la carga de trabajo. Por omisión, la habilitación del equilibrado de la carga de trabajo de Sysplex habilita el redireccionamiento de cliente automático. Establezca otras propiedades para las que se considere que los valores por omisión no son aceptables. Por ejemplo, el código siguiente establece los valores de las propiedades que se indican en Tabla 47 en la página 310.

   ```
   java.util.Properties properties = new java.util.Properties();
   properties.put("user", "xxxx");
   properties.put("password", "yyyy");
   properties.put("enableSysplexWLB", "true");
   properties.put("maxTransportObjects", "80");
   properties.put("maxRetriesForClientReroute", "10");
   properties.put("retryIntervalForClientReroute", "20");
   java.sql.Connection con =
     java.sql.DriverManager.getConnection(url, properties);
   ```

3. Establezca las propiedades de configuración de IBM Data Server Driver para JDBC y SQLJ para refinar el equilibrado de la carga de trabajo para todas las instancias de DataSource o Connection que se han creado bajo el controlador.

Establezca las propiedades de configuración en un archivo DB2JccConfiguration.properties siguiendo estos pasos:

a. Cree un archivo DB2JccConfiguration.properties o edite el archivo DB2JccConfiguration.properties existente.
b. Establezca la propiedad de configuración db2.jcc.maxTransportObjects sólo si se definen varios objetos DataSource que se refieran al mismo grupo de compartimiento de datos, y deba limitarse el número de conexiones en los diferentes objetos DataSource.

   Comience con un valor similar al siguiente:

   ```
   db2.jcc.maxTransportObjects=500
   ```
c. Incluya el directorio que contiene DB2JccConfiguration.properties en la concatenación CLASSPATH.

Funcionamiento del equilibrado de la carga de trabajo de Sysplex para conexiones de clientes de Java a servidores DB2 para z/OS

El equilibrado de la carga de trabajo de Sysplex (también llamado equilibrado de la carga de trabajo en el nivel de transacción) para las conexiones con DB2 para z/OS contribuye a conseguir una alta disponibilidad mediante el equilibrado de trabajo entre miembros de un grupo de compartimiento de datos al comienzo de una transacción.

En la visión general siguiente se describen los pasos que tienen lugar cuando un cliente se conecta a un Sysplex de DB2 para z/OS y se habilita el equilibrado de la carga de trabajo de Sysplex:

1. Cuando el cliente establece una conexión por primera vez mediante la dirección IP de todo el sysplex, conocida como la dirección IP de grupo, o cuando otro objeto de conexión reutiliza una conexión, el servidor devuelve información de distribución de cargas de trabajo de miembros.

   La vida útil por omisión de la lista de servidores en la memoria caché es de 30 segundos.
2. Al inicio de una transacción nueva, el cliente lee la lista de servidores almacenada en la antememoria para identificar a un miembro que tenga capacidad no intervenida y busca en la agrupación de transporte un transporte desocupado que esté vinculado al miembro infrautilizado. (Un transporte desocupado es un transporte que no tiene ningún objeto de conexión asociado).
   - Si un transporte desocupado está disponible, el cliente asocia el objeto de conexión con el transporte.
   - Si, después de un tiempo de espera excedido configurable por el usuario, no hay ningún transporte desocupado disponible en la agrupación de transporte y no puede asignarse ningún transporte nuevo puesto que la agrupación de transporte ha alcanzado su límite, se devuelve un error a la aplicación.
3. Cuando la transacción se ejecuta, accede al miembro que está vinculado al transporte.

   Cuando la primera sentencia de SQL en una transacción se ejecuta, si IBM Data Server Driver para JDBC y SQLJ recibe un error de comunicación porque el servidor de datos desactiva la conexión o se sobrepasa el valor blockingReadConnectionTimeout, el controlador vuelve a intentar la sentencia de SQL 10 veces antes de notificar un error. En cada reintento, el controlador cierra el transporte existente, obtiene un nuevo transporte y, a continuación, ejecuta la transacción. Durante estos reintentos, si las propiedades

maxRetriesForClientReroute y retryIntervalForClientReroute están establecidas, sus valores sólo se aplican al proceso de obtener un nuevo transporte durante cada reintento.

4. Cuando la transacción finaliza, el cliente verifica con el servidor que la reutilización del transporte aún está permitida para el objeto de conexión.
5. Si la reutilización del transporte aún está permitida, el servidor devolverá una lista de sentencias SET para registros especiales que se apliquen al entorno de ejecución para el objeto de conexión.

   El cliente almacena en la antememoria estas sentencias, que reproduce con el fin de reconstruir el entorno de ejecución cuando el objeto de conexión está asociado con un transporte nuevo.
6. El objeto de conexión se desasocia después del transporte.
7. La copia de cliente de la lista de servidores se renueva cuando se establece una nueva conexión, o cada 30 segundos.
8. Cuando el equilibrado de la carga de trabajo es obligatorio para una transacción nueva, el cliente utiliza el mismo proceso para asociar el objeto de conexión con un transporte.

Funcionamiento del redireccionamiento de cliente automático para conexiones de clientes de Java a DB2 para z/OS

El soporte del redireccionamiento de cliente automático proporciona soporte de migración tras error si un IBM Data Server Client pierde la conectividad con un miembro de un Sysplex de DB2 para z/OS. El redireccionamiento automático de clientes permite al cliente recuperarse de una anomalía intentando reconectarse a la base de datos a través de cualquier miembro disponible de un Sysplex.

El redireccionamiento de cliente automático está habilitado por omisión si se ha habilitado el equilibrado de la carga de trabajo de Sysplex.

El soporte de cliente para el redireccionamiento de cliente automático está disponible en los clientes del servidor de datos de IBM que disponen de una licencia de DB2 Connect. No es necesario el servidor de DB2 Connect para llevar a cabo el redireccionamiento de cliente automático.

El redireccionamiento de cliente automático para DB2 para z/OS funciona de este modo:

1. Como parte de la respuesta a una petición COMMIT del cliente, el servidor de datos devuelve:
   - Un indicador que especifica si se pueden volver a usar los transportes. Los transportes se pueden volver a usar si no quedan recursos, como cursores retenidos.
   - Sentencias SET que el cliente puede usar para reproducir el estado de la conexión durante la reutilización del transporte.
2. Si la primera sentencia de SQL de una transacción falla, y el transporte se puede volver a usar:
   - No se informa de ningún error a la aplicación.
   - Se ejecuta de nuevo la sentencia de SQL anómala.
   - Las sentencias SET que están asociadas a la conexión lógica se vuelven a reproducir para restaurar el estado de conexión.
3. Si una sentencia de SQL que no sea la primera sentencia de SQL de una transacción falla, y los transportes se pueden volver a usar:

- La transacción se ha retrotraído.
- La aplicación se vuelve a conectar con el servidor de datos.
- Las sentencias SET que están asociadas a la conexión lógica se vuelven a reproducir para restaurar el estado de conexión.
- Se devuelve el error SQL -30108 (para Java) o SQL30108N (para clientes que no son de Java) a la aplicación para avisar de la retrotracción y la reconexión satisfactoria. La aplicación debe incluir el código para volver a intentar la transacción anómala.

4. Si una sentencia de SQL que no sea la primera sentencia de SQL de una transacción falla, y los transportes no se pueden volver a usar:
   - La conexión lógica volverá a su estado inicial, por omisión.
   - Se devuelve el error SQL -30081 (para Java) o SQL30081N (para clientes que no son de Java) a la aplicación para avisar que la reconexión no fue satisfactoria. La aplicación debe reconectarse al servidor de datos, restablecer el estado de conexión y reintentar la transacción anómala.
5. Si se han intentado las conexiones con todos los miembros de la lista de miembros de compartimiento de datos, y ninguna ha tenido éxito, se intenta una conexión mediante el URL que esté asociado con el grupo de compartimiento de datos, para determinar si hay miembros disponibles.

Funcionamiento del soporte de grupos alternativos

El soporte de grupos alternativos permite que IBM Data Server Driver para JDBC y SQLJ mueva una carga de trabajo de aplicaciones a un grupo de compartimiento de datos alternativo cuando el grupo de compartimiento de datos primario no está disponible.

Recomendación: Para el soporte de grupos alternativos, los grupos de compartimiento de datos primarios y alternativos deben utilizar la dirección IP de Sysplex distribuido que recibe el soporte del distribuidor de sysplex en z/OS.

Importante: Si está utilizando la versión de IBM Data Server Driver para JDBC y SQLJ que se proporciona con DB2 9.7 Fixpack 6, debe aplicar el APAR IC79084 para que el soporte de grupos alternativos esté disponible en el sistema.

Para habilitar el soporte de grupos alternativos, proporcione las direcciones de grupos de compartimiento de datos alternativos en las propiedades alternateGroupServerName, alternateGroupPortNumber y alternateGroupDatabaseName de Connection o DataSource.

Asimismo, puede controlar si el comportamiento de la migración tras error sin fisuras ha de estar o no en vigor para el soporte de grupos alternativos estableciendo la propiedad enableAlternateGroupSeamlessACR de Connection o DataSource.

Para que el funcionamiento del soporte de grupos alternativos sea correcto, los datos del grupo primario y del grupo alternativo deben ser los mismos.

Tras una migración tras error desde el grupo primario hasta el grupo alternativo, el valor de la propiedad databaseName no cambia.

La migración tras error sin fisuras de grupo alternativo permite realizar la migración tras error sin fisuras únicamente desde el grupo primaria a un grupo alternativo. Después de la migración tras error sin fisuras, todas las conexiones en una instancia de DataSource se establecen con el grupo alternativo. DataSource no

puede crear conexiones de vuelta al grupo primario, aunque el grupo primario pase a estar disponible y todas las conexiones existentes con el grupo alternativo se hayan cerrado. Después de que las conexiones en una instancia de DataSource se hayan movido al grupo alternativo, la única forma de asociar estas conexiones con el grupo primario es reciclando el entorno de ejecución de Java (JVM). Si una instancia de DataSource se está ejecutando dentro de Websphere Application Server, se debe reciclar el servidor de aplicaciones completo para mover las conexiones al grupo primario.

Después de la migración tras error sin fisuras, si una nueva instancia de DataSource se instancia mediante una aplicación dentro de la misma JVM desde la que anteriormente las conexiones han realizado una migración tras error sin fisuras a un servidor alternativo, IBM Data Server Driver para JDBC y SQLJ permite establecer conexiones con el grupo primario cuando el grupo primario pase a estar disponible, incluso si otras conexiones de DataSource que se ejecutan dentro de la misma JVM debe conectar con el grupo alternativo.

Si una conexión que creó mediante DriverManager.getConnection realiza una migración tras error sin fisuras a un grupo alternativo, todas las conexiones posteriores que se obtienen a través de DriverManager.getConnection y tienen el mismo URL y las mismas propiedades también se conectan al grupo alternativo, incluso si el grupo primario pasa a estar disponible. La única forma de mover una conexión al grupo primario con DriverManager.getConnection es creando una conexión con URL o propiedades diferentes.

El soporte de grupos alternativos funciona del modo siguiente:

- Para la primera conexión de una aplicación con un grupo de compartimiento de datos:
  1. IBM Data Server Driver para JDBC y SQLJ intenta conectar la aplicación con el grupo de compartimiento de datos primario.
  2. Si la conexión falla, el controlador intenta conectar la aplicación con el grupo de compartimiento de datos alternativo que se especifica mediante el conjunto de valores de las propiedades alternateGroupServerName, alternateGroupPortNumber y alternateGroupDatabaseName.
  3. Si se establece la conexión con el grupo de compartimiento de datos alternativo, no se enviará ningún error de SQL a la aplicación si enableAlternateGroupSeamlessACR se ha establecido en `true`. De otro modo, se devolverá el error de SQL -30108 a la aplicación. El siguiente uso de la conexión establecerá la conexión con el grupo alternativo.

     Si no se establece la conexión con el grupo de compartimiento de datos alternativo, el controlador devuelve el error de SQL -4499 a la aplicación.
- Para una conexión posterior, tras haberse conectado la aplicación con el grupo de compartimiento de datos primario:
  1. IBM Data Server Driver para JDBC y SQLJ intenta conectar la aplicación con cada uno de los miembros disponibles del grupo de compartimiento de datos primario.
  2. Si ninguno de los miembros del grupo de compartimiento de datos primario la está disponible al primer intento, el controlador vuelve a intentar la conexión con el grupo de compartimiento de datos primario utilizando la dirección que especifica el conjunto de valores de las propiedades serverName, portNumber y databaseName de Connection o DataSource.
  3. Si la conexión con el grupo primario falla, el controlador intenta conectar la aplicación con el grupo de compartimiento de datos alternativo que se

especifica mediante el conjunto de valores de las propiedades alternateGroupServerName, alternateGroupPortNumber y alternateGroupDatabaseName.

4. Si no se establece una conexión, el controlador devuelve un error de SQL -4499 a la aplicación.

- Para una conexión posterior, tras haberse conectado la aplicación con el grupo de compartimiento de datos alternativo:
  1. IBM Data Server Driver para JDBC y SQLJ intenta conectar la aplicación con cada uno de los miembros disponibles del grupo de compartimiento de datos alternativo.
  2. Si no se establece una conexión, el controlador devuelve un error de SQL -4499 a la aplicación.
- Para una conexión con un grupo de compartimiento de datos primario que se encuentra en una transacción:
  1. IBM Data Server Driver para JDBC y SQLJ intenta conectar la aplicación con el grupo de compartimiento de datos alternativo.
  2. Si se establece una conexión, enableAlternateSeamlessGroupACR se ha establecido en `true` y la transacción es apta para la migración tras error sin fisuras, vuelve a intentarse la transacción.
  3. Si se establece una conexión, enableAlternateSeamlessGroupACR se ha establecido en `true` y la transacción no es apta para la migración tras error sin fisuras, el controlador devuelve el error de SQL -30108 a la aplicación.
  4. Si se establece una conexión y enableAlternateSeamlessGroupACR se ha establecido en `false`, el controlador devuelve el error de SQL -30108 a la aplicación.
  5. Si no se establece una conexión, el controlador devuelve un error de SQL -4499 a la aplicación.

Ejemplos

Supongamos que se han definido dos grupos de compartimiento de datos: PG1 y AG1. Ambos utilizan una dirección IP de grupo de DB2 para z/OS. PG1 es el grupo de compartimiento de datos primario y AG1 es el grupo de compartimiento de datos alternativo para el soporte de grupos de compartimiento de datos alternativos de IBM Data Server Driver para JDBC y SQLJ.

Supongamos que los grupos de compartimiento de datos tienen los valores de servidor, puerto y base de datos siguientes:

Grupo de compartimiento de datos	Valores de servidor, puerto y base de datos
PG1	host1, port1, dbname1
AG1	host2, port2, dbname2

Supongamos también que se han establecido los valores de propiedad siguientes:

Propiedad	Valor
serverName	host1
portNumber	port1
databaseName	dbname1
alternateGroupServerName	host2

Propiedad	Valor
alternateGroupPortNumber	port2
alternateGroupDatabaseName	dbname2
enableAlternateGroupSeamlessACR	true

En los pasos siguientes se ofrece una demostración de una situación de ejemplo de grupo de compartimiento de datos alternativo para una conexión con PG1 que falla:

1. El controlador intenta conectar la aplicación con PG1 utilizando host1:port1.
2. La conexión falla.
3. El controlador intenta conectar la aplicación con AG1 utilizando host2:port2.
4. La conexión se establece satisfactoriamente.
5. La aplicación sigue ejecutándose.
6. Todos los miembros de AG1 dejan de estar disponibles y la conexión con AG1 falla.
7. El controlador emite el error de SQL -4499.

En los pasos siguientes se ofrece una demostración de una situación de ejemplo de grupo alternativo para una conexión con PG1 que no se ejecuta correctamente durante una transacción:

1. El controlador intenta conectar la aplicación con PG1 utilizando host1:port1.
2. La conexión es correcta.
3. La aplicación inicia la ejecución del trabajo.
4. Todos los miembros de PG1 caen.
5. El controlador intenta conectar la aplicación con AG1 utilizando host2:port2.
6. La conexión se establece satisfactoriamente.
7. La aplicación cumple los criterios para la migración tras error sin fisuras, por lo tanto, vuelve a intentarse la transacción.
8. El reintento no se ejecuta correctamente.
9. El controlador emite el error de SQL -30108 y retrotrae el trabajo hasta el punto de confirmación anterior.

Requisitos de programación de aplicaciones para obtener una alta disponibilidad para conexiones de clientes de Java con servidores DB2 para z/OS

La migración tras error para el redireccionamiento de cliente automático puede ser con o sin fisuras. Si la migración tras error para las conexiones con DB2 para z/OS no se produce sin fisuras, debe agregar el código para tener en cuenta los errores que se han devuelto cuando se produce la migración tras error.

Si la migración tras error no se produce sin fisuras y se vuelve a establecer una conexión con el servidor, se devuelve SQLCODE -30108 (SQL30108N) a la aplicación. Todo el trabajo que se ha producido en la transacción actual se ha retrotraído. En la aplicación, necesita:

- Comprobar el código de razón que se devuelve con el error -30108 para determinar si los valores de registro especial que han tenido lugar desde el miembro de compartimiento de datos que experimenta la anomalía hasta el nuevo miembro de compartimiento de datos (migración tras error) eran los

valores correspondientes al punto de confirmación más reciente o los valores correspondientes al punto de anomalía. Restaure cualquier valor de registro especial que no sea actual.

- Ejecutar todas las operaciones de SQL que han tenido lugar desde la operación de confirmación anterior.

Se deben cumplir las condiciones siguientes para que se produzca la migración tras error sin fisuras, para las conexiones directas con DB2 para z/OS:

- El lenguaje de la aplicación es Java, CLI o .NET.
- La conexión no está en la transacción. Es decir, la anomalía se produce cuando la primera sentencia de SQL se ejecuta en la transacción.
- El servidor de datos permite la reutilización del transporte al final de la transacción anterior. Se produce una excepción a esta condición si la reutilización del transporte no queda garantizada porque la aplicación se vinculó con KEEPDYNAMIC(YES).
- Todos los datos de sesiones globales se han cerrado o descartado.
- No hay cursores retenidos abiertos.
- Si la aplicación utiliza CLI, ésta no podrá llevar a cabo acciones que requieran al controlador que mantenga un historial de las API a las que se ha llamado anteriormente, con el fin de reproducir la sentencia de SQL. Ejemplos de dichas acciones son la especificación de datos durante la ejecución, ejecutar sentencias de SQL compuesto o utilizar entradas de matrices.
- La aplicación no es un procedimiento almacenado.
- La aplicación no se ejecuta en un entorno federado.
- Se utiliza la confirmación en dos fases, si las transacciones dependen del éxito de las transacciones anteriores. Si se produce una anomalía durante una operación de confirmación, el cliente no tiene información sobre si el trabajo se confirmó o se retrotrajo al servidor. Si cada transacción depende del éxito de la transacción anterior, utilice una confirmación en dos fases. La confirmación en dos fases requiere el uso del soporte para XA.

Capítulo 10. Java 2 Platform, Enterprise Edition

Java 2 Platform Enterprise Edition (J2EE) reduce el coste y la complejidad de desarrollar servicios de varios niveles.

En el entorno empresarial global actual, las organizaciones tienen que ampliar su alcance, reducir sus costes y reducir sus tiempos de respuesta, proporcionando servicios a los que puedan acceder fácilmente sus clientes, empleados, proveedores y otros socios empresariales. Estos servicios deben tener las siguientes características:

- Altamente disponibles, para ajustarse a los requisitos del entorno empresarial global
- Seguros, para proteger la privacidad de los usuarios y la integridad de la empresa
- Fiables y escalables, de modo que las transacciones empresariales resulten precisas y se procesen con rapidez

En la mayoría de los casos, estos servicios se suministran con la ayuda de aplicaciones de varios enlaces en las que cada enlace tiene un objetivo específico.

J2EE consigue estas ventajas definiendo una arquitectura estándar que se suministra como los siguientes elementos:

- J2EE Application Model, un modelo de aplicación estándar para desarrollar servicios de cliente ligero de varios niveles
- J2EE Platform, una plataforma estándar para albergar aplicaciones de J2EE
- J2EE Compatibility Test Suite para verificar que un producto de la plataforma J2EE cumple con el estándar de dicha plataforma
- J2EE Reference Implementation para demostrar las funciones de J2EE y para proporcionar una definición operativa de la plataforma J2EE

Soporte para componentes de aplicación Java 2 Platform, Enterprise Edition

Java 2 Platform Enterprise Edition (J2EE) proporciona el entorno de tiempo de ejecución para alojar aplicaciones J2EE.

El entorno de tiempo de ejecución define cuatro tipos de componentes de aplicación a los que un producto J2EE debe dar soporte:

- Los clientes de aplicaciones son programas de lenguaje de programación Java que suelen ser programas GUI que se ejecutan en un sistema de escritorio. Los clientes de aplicaciones tienen acceso a todas las funciones del enlace medio de J2EE.
- Los componentes applets y GUI que normalmente se ejecutan en un navegador web pero que se pueden ejecutar en otras aplicaciones o dispositivos que den soporte al modelo de programación de applets.
- Los servlets, JavaServer Pages (JSP), filtros y receptores de sucesos de la web que se suelen ejecutar en un navegador web y que pueden responder a peticiones HTTP procedentes de clientes web. Los servlets, JSP y filtros se pueden utilizar para generar páginas HTML que constituyen la interfaz de usuario de una aplicación. También se pueden utilizar para generar XML o datos en otro

formato que consumen otros componentes de la aplicación. Los servlets, las páginas creadas con tecnología JSP, los filtros y los receptores de sucesos de la web reciben conjuntamente en esta especificación el nombre *componentes de la web*. Las aplicaciones web constan de componentes de la web y de otros datos como páginas HTML.

- Los componentes Enterprise JavaBeans (EJB) se ejecutan en un entorno gestionado que da soporte a transacciones. Los Enterprise Beans suelen contener la lógica empresarial correspondiente a una aplicación J2EE.

Los componentes de aplicaciones listados anteriormente se pueden dividir en tres categorías, según el modo en que se pueden desplegar y gestionar:

- Componentes que se despliegan, gestionan y ejecutan en un servidor J2EE.
- Componentes que se despliegan y gestionan en un servidor J2EE pero que se cargan en una máquina cliente y se ejecutan en la misma.
- Componentes cuyo despliegue y gestión no están completamente definidos por esta especificación. Los clientes de aplicaciones pueden encontrarse en esta categoría.

El soporte de tiempo de ejecución correspondiente a estos componentes se proporciona mediante *contenedores*.

Contenedores de Java 2 Platform, Enterprise Edition

Un contenedor proporciona una vista federada de las API subyacentes de Java 2 Platform Enterprise Edition (J2EE) a los componentes de la aplicación.

Un producto J2EE típico proporcionará un contenedor para cada tipo de componente de aplicación: contenedor de clientes de la aplicación, contenedor de applets, contenedor de web y contenedor de Enterprise Beans. Las herramientas de contenedor también comprenden los formatos de archivo para empaquetar los componentes de la aplicación para su despliegue.

La especificación necesita que estos contenedores proporcionen un entorno de tiempo de ejecución compatible con Java. Esta especificación define un conjunto de servicios estándares a los que debe dar soporte cada producto J2EE. Estos servicios estándar son:

- Servicio HTTP
- Servicio HTTPS
- API de transacciones Java
- Método de invocación remota
- IDL Java
- API JDBC
- Servicio de mensajes de Java
- Java Naming and Directory Interface
- JavaMail
- Infraestructura de activación de JavaBeans
- API Java para el análisis XML
- Arquitectura de conectores
- Servicio de autentificación y autorización de Java

Servidor Java 2 Platform, Enterprise Edition

Una parte de un contenedor Java 2 Platform Enterprise Edition (J2EE) es un servidor.

Normalmente, un proveedor de productos J2EE implementa la funcionalidad de lado del servidor de J2EE, mientras que la funcionalidad de cliente de J2EE se crea en tecnología J2SE.

IBM WebSphere Application Server es un servidor que cumple las especificaciones de J2EE.

Requisitos de la base de datos de Java 2 Platform, Enterprise Edition

Java 2 Platform Enterprise Edition necesita un servidor de datos para almacenar los datos de la empresa. Debe poder accederse al servidor de datos mediante la API de JDBC.

Se puede acceder a la base de datos desde componentes de la web, Enterprise Beans y componentes cliente de la aplicación. No hace falta que se pueda acceder a la base de datos desde los applets.

Java Naming and Directory Interface (JNDI)

JNDI permite que las aplicaciones basadas en la plataforma Java accedan a varios servicios de asignación de nombres y de directorio.

Forma parte del conjunto de interfaces de programación de aplicaciones (API) de Java Enterprise. JNDI permite que los desarrolladores de aplicaciones creen aplicaciones portables que están habilitadas para varios servicios de nombres y de directorio, tales como sistemas de archivos; servicios de directorio tales como Lightweight Directory Access Protocol (LDAP) y Novell Directory Services, y sistemas de objetos distribuidos tales como Common Object Request Broker Architecture (CORBA), Java Remote Method Invocation (RMI), y Enterprise JavaBeans (EJB).

La API JNDI tiene dos partes: una interfaz de nivel de aplicación que utilizan los componentes de la aplicación para acceder a los servicios de nomenclatura y directorio y una interfaz de proveedor de servicios para conectar con un proveedor de un servicio de nomenclatura y directorio.

Gestión de transacciones Java

Java 2 Platform Enterprise Edition (J2EE) simplifica la programación de aplicaciones para la gestión de transacciones distribuidas.

J2EE incluye soporte para transacciones distribuidas a través de dos especificaciones, API de transacciones Java (JTA) y Servicio de transacciones Java (JTS). JTA es una API de alto nivel, independiente de la implementación e independiente del protocolo, que permite a las aplicaciones y a los servidores de aplicaciones acceder a transacciones. Además, JTA está siempre habilitada.

IBM Data Server Driver para JDBC y SQLJ aplica las especificaciones JTA y JTS.

Para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4, se da soporte a las transacciones distribuidas en los servidores DB2 Database para Linux, UNIX y Windows, DB2 para z/OS y DB2 para i.

JTA especifica interfaces Java estándares entre un gestor de transacciones y las partes que intervienen en un sistema de transacciones distribuidas: el gestor de recursos, el servidor de aplicaciones y las aplicaciones transaccionales.

JTS especifica la implementación de un Gestor de transacciones que da soporte a JTA e implementa la correlación Java de la especificación OMG Object Transaction Service (OTS) 1.1 al nivel que hay bajo la API. JTS propaga transacciones mediante IIOP.

JTA y JTS permiten que los servidores J2EE de aplicaciones eviten al desarrollador de componentes las tareas de gestión de transacciones. Los programadores pueden definir las propiedades transaccionales de la tecnología EJB basándose en componentes durante el diseño o despliegue mediante sentencias declarativas en el descriptor de despliegue. El servidor de aplicaciones se hace cargo de la responsabilidad de la gestión de transacciones.

En el entorno de DB2 y WebSphere Application Server, WebSphere Application Server asume el rol de gestor de transacciones, y DB2 actúa como gestor de recursos. WebSphere Application Server implementa JTS y parte de JTA, y los controladores JDBC también implementan parte de JTA, por lo que WebSphere Application Server y DB2 pueden proporcionar transacciones distribuidas coordinadas.

No es necesario configurar DB2 para que esté habilitado para JTA en el entorno de WebSphere Application Server, pues los controladores JDBC detectan automáticamente este entorno.

IBM Data Server Driver para JDBC y SQLJ proporciona estas dos clases DataSource:
- com.ibm.db2.jcc.DB2ConnectionPoolDataSource
- com.ibm.db2.jcc.DB2XADataSource

WebSphere Application Server proporciona conexiones de uso compartido para bases de datos. Si la aplicación intervendrá en una transacción distribuida, se debe utilizar la clase com.ibm.db2.jdbc.DB2XADataSource al definir fuentes de datos DB2 dentro del WebSphere Application Server.

Para conocer información detallada sobre cómo configurar el WebSphere Application Server con DB2, consulte el centro de información de WebSphere Application Server, situado en:

`http://www.ibm.com/software/webservers/appserv/library.html`

Ejemplo de una transacción distribuida que utiliza métodos de JTA

Normalmente las transacciones distribuidas suponen varias conexiones con una misma fuente de datos o con fuentes de datos diferentes, que pueden ser de fabricantes diversos.

La mejor forma de mostrar la utilización de transacciones distribuidas es compararlas con transacciones locales. En las transacciones locales, una aplicación JDBC confirma los cambios hechos en una base de datos e indica el final de una unidad de trabajo en una de las formas siguientes:
- Invocando los métodos Connection.commit o Connection.rollback después de ejecutar una o más sentencias de SQL
- Invocando el método Connection.setAutoCommit(true) al inicio de la aplicación para que los cambios se confirmen después de cada sentencia de SQL

La Figura 48 muestra el código mediante el que se ejecutan transacciones locales.

```
con1.setAutoCommit(false);  // Desactivar la confirmación automática
// Ejecutar sentencias de SQL
...
con1.commit();              // Confirmar la transacción
// Ejecutar más sentencias de SQL
...
con1.rollback();            // Retrotraer la transacción
con1.setAutoCommit(true);   // Permitir la confirmación después
                            // de cada sentencia de SQL
...
// Ejecutar más sentencias de SQL, que se confirmarán
// automáticamente después de cada sentencia de SQL.
```

Figura 48. Ejemplo de transacción local

En cambio, las aplicaciones que intervienen en transacciones distribuidas no pueden invocar los métodos Connection.commit, Connection.rollback, ni Connection.setAutoCommit(true) dentro de la transacción distribuida. En las transacciones distribuidas, los métodos Connection.commit o Connection.rollback no indican límites de transacción. En lugar de ello, las aplicaciones dejan que el servidor de aplicaciones gestione los límites de transacción.

La Figura 49 muestra una aplicación que utiliza transacciones distribuidas. Mientras se ejecuta el código mostrado en el ejemplo, el servidor de aplicaciones también está ejecutando otros EJB que forman parte de la misma transacción distribuida. Cuando todos los EJB han invocado utx.commit(), el servidor de aplicaciones confirma la transacción distribuida completa. Si cualquiera de los EJB no se ejecuta satisfactoriamente, el servidor de aplicaciones retrotrae todo el trabajo hecho por todos los EJB que están asociados a la transacción distribuida.

```
javax.transaction.UserTransaction utx;
// Utilizar el método begin sobre un objeto UserTransaction
// para indicar el inicio de una transacción distribuida.
utx.begin();
...
// Ejecute sentencias de SQL con un objeto Connection.
// No invoque los métodos commit ni rollback de Connection.
...
// Utilizar el método commit sobre el objeto UserTransaction
// para hacer que se confirmen todas las ramas de transacción
// e indicar el final de la transacción distribuida.

utx.commit();
...
```

Figura 49. Ejemplo de transacción distribuida cuando se utiliza un servidor de aplicaciones

La Figura 50 muestra un programa que utiliza métodos de JTA para ejecutar una transacción distribuida. Este programa actúa como gestor de transacciones y aplicación transaccional. Dos conexiones con dos fuentes de datos diferentes ejecutan tareas de SQL dentro de una sola transacción distribuida.

Figura 50. Ejemplo de transacción distribuida que hace uso de la JTA

```
class XASample
{
  javax.sql.XADataSource xaDS1;
  javax.sql.XADataSource xaDS2;
  javax.sql.XAConnection xaconn1;
```

```
javax.sql.XAConnection xaconn2;
javax.transaction.xa.XAResource xares1;
javax.transaction.xa.XAResource xares2;
java.sql.Connection conn1;
java.sql.Connection conn2;

public static void main (String args []) throws java.sql.SQLException
{
  XASample xat = new XASample();
  xat.runThis(args);
}
// En calidad de gestor de transacciones, este programa proporciona
// el ID de transacción global y el calificador de rama. El ID de
// transacción global y el calificador de rama no deben ser iguales
// entre sí, y la combinación formada por ambos debe ser exclusiva
// para este gestor de transacciones.
public void runThis(String[] args)
{
  byte[] gtrid = new byte[] { 0x44, 0x11, 0x55, 0x66 };
  byte[] bqual = new byte[] { 0x00, 0x22, 0x00 };
  int rc1 = 0;
  int rc2 = 0;

  try
  {

    javax.naming.InitialContext context = new javax.naming.InitialContext();
    /*
      * Observe que se usa javax.sql.XADataSource en lugar de una implementación
      * de controlador específica tal como com.ibm.db2.jcc.DB2XADataSource.
      */
    xaDS1 = (javax.sql.XADataSource)context.lookup("checkingAccounts");
    xaDS2 = (javax.sql.XADataSource)context.lookup("savingsAccounts");

    // XADatasource contiene el ID de usuario y la contraseña.
    // Obtener el objeto XAConnection de cada XADataSource
    xaconn1 = xaDS1.getXAConnection();
    xaconn2 = xaDS2.getXAConnection();

    // Obtener el objeto java.sql.Connection de cada XAConnection
    conn1 = xaconn1.getConnection();
    conn2 = xaconn2.getConnection();

    // Obtener el objeto XAResource de cada XAConnection
    xares1 = xaconn1.getXAResource();
    xares2 = xaconn2.getXAResource();
    // Crear el objeto Xid para esta transacción distribuida.
    // Este ejemplo utiliza la implementación com.ibm.db2.jcc.DB2Xid
    // de la interfaz Xid. Xid se puede utilizar con cualquier controlador JDBC
    // que da soporte a JTA.
    javax.transaction.xa.Xid xid1 =
                            new com.ibm.db2.jcc.DB2Xid(100, gtrid, bqual);

    // Inicie la transacción distribuida en las dos conexiones.
    // NO es necesario iniciar y finalizar las dos conexiones juntas.
    // Esto puede hacerse en hebras diferentes, junto con sus operaciones de SQL.
    xares1.start(xid1, javax.transaction.xa.XAResource.TMNOFLAGS);
    xares2.start(xid1, javax.transaction.xa.XAResource.TMNOFLAGS);
...
    // Ejecute las operaciones de SQL en la conexión 1.
    // Ejecute las operaciones de SQL en la conexión 2.
...
    // Ahora finalice la transacción distribuida en las dos conexiones.
    xares1.end(xid1, javax.transaction.xa.XAResource.TMSUCCESS);
    xares2.end(xid1, javax.transaction.xa.XAResource.TMSUCCESS);

    // Si el trabajo de la conexión 2 se ha realizado en otra hebra,
```

```
// es necesaria aquí una llamada a thread.join() para esperar
// a que termine el trabajo de la conexión 2.

try
{ // Ahora prepare ambas ramas de la transacción distribuida.
  // Ambas ramas se deben preparar satisfactoriamente para
  // poder confirmar los cambios.
  // Si la transacción distribuida falla, se emite una
  // excepción XAException.
  rc1 = xares1.prepare(xid1);
  if(rc1 == javax.transaction.xa.XAResource.XA_OK)
  { // La preparación fue satisfactoria. Prepare la segunda conexión
    rc2 = xares2.prepare(xid1);
    if(rc2 == javax.transaction.xa.XAResource.XA_OK)
    { // Ambas conexiones se prepararon satisfactoriamente
      // y ninguna de ella era de solo lectura.
      xares1.commit(xid1, false);
      xares2.commit(xid1, false);
    }
    else if(rc2 == javax.transaction.xa.XAException.XA_RDONLY)
    { // La segunda conexión es de solo lectura, por lo que
      // solo se confirma la primera conexión.
      xares1.commit(xid1, false);
    }
  }
  else if(rc1 == javax.transaction.xa.XAException.XA_RDONLY)
  { // El SQL de la primera conexión es de solo lectura
    // (tal como un SELECT).
    // La preparación ha confirmado la conexión. Prepare la
    // segunda conexión.
    rc2 = xares2.prepare(xid1);
    if(rc2 == javax.transaction.xa.XAResource.XA_OK)
    { // La primera conexión es de solo lectura, pero la
      // segunda no lo es.
      // Confirme la segunda conexión.
      xares2.commit(xid1, false);
    }
    else if(rc2 == javax.transaction.xa.XAException.XA_RDONLY)
    { // Ambas conexiones son de solo lectura, y ambas
      // están ya confirmadas, por lo que no es necesaria
      // ninguna otra acción.
    }
  }
}      catch (javax.transaction.xa.XAException xae)
{ // La transacción distribuida ha fallado,
  // por lo que debe retrotraerla.
  // Notificar XAException para preparación/confirmación.
  System.out.println("Distributed transaction prepare/commit failed. " +
                     "Rolling it back.");
  System.out.println("XAException error code = " + xae.errorCode);
  System.out.println("XAException message = " + xae.getMessage());
  xae.printStackTrace();
  try
  {
    xares1.rollback(xid1);
  }
  catch (javax.transaction.xa.XAException xae1)
  { // Notificar error de la retrotracción.
    System.out.println("distributed Transaction rollback xares1 failed");
    System.out.println("XAException error code = " + xae1.errorCode);
    System.out.println("XAException message = " + xae1.getMessage());
  }
  try
  {
    xares2.rollback(xid1);
  }
  catch (javax.transaction.xa.XAException xae2)
```

```
        { // Notificar error de la retrotracción.
          System.out.println("distributed Transaction rollback xares2 failed");
          System.out.println("XAException error code = " + xae2.errorCode);
          System.out.println("XAException message = " + xae2.getMessage());
        }
      }

      try
      {
        conn1.close();
        xaconn1.close();
      }
      catch (Exception e)
      {
        System.out.println("Failed to close connection 1: " + e.toString());
        e.printStackTrace();
      }
      try
      {
        conn2.close();
        xaconn2.close();
      }
      catch (Exception e)
      {
        System.out.println("Failed to close connection 2: " + e.toString());
        e.printStackTrace();
      }
    }
    catch (java.sql.SQLException sqe)
    {
      System.out.println("SQLException caught: " + sqe.getMessage());
      sqe.printStackTrace();
    }
    catch (javax.transaction.xa.XAException xae)
    {
      System.out.println("XA error is " + xae.getMessage());
      xae.printStackTrace();
    }
    catch (javax.naming.NamingException nme)
    {
      System.out.println(" Naming Exception: " + nme.getMessage());
    }
  }
}
```

Recomendación: Para lograr un mejor rendimiento, finalice una transacción distribuida antes de iniciar otra transacción distribuida o local.

Establecimiento del valor de tiempo excedido de transacción para una instancia de XAResource

Utilice el método XAResource.setTransactionTimeout para reducir la aparición de puntos muertos en una base de datos DB2 que es el destino de transacciones distribuidas.

Acerca de esta tarea

Una transacción distribuida destinada a DB2 Database para Linux, UNIX y Windows que finaliza, pero no se puede preparar, no es una transacción dudosa. Por tanto, el gestor de transacciones no puede recuperar la transacción, y el gestor de recursos de DB2 no coloca la transacción en su lista de transacciones dudosas. El gestor de recursos de DB2 no retrotrae la transacción inmediatamente, sino que espera a que se liberen todas las conexiones con la base de datos. Durante este

periodo de inactividad, la transacción sigue manteniendo bloqueos en la base de datos. Si el gestor de transacciones no desconecta todas las conexiones con la base de datos para permitir la retrotracción, la transacción finalizada sigue manteniendo el bloqueo de registros de la base de datos. Si otra aplicación intenta acceder a esos registros bloqueados, se puede producir un punto muerto.

En una aplicación Java que utiliza transacciones distribuidas e IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4, puede evitar que una transacción mantenga indefinidamente bloqueos sobre una base de datos. Para ello invoque el método XAResource.setTransactionTimeout para establecer un valor de tiempo de espera para las transacciones. Para hacer esto, siga estos pasos:

Procedimiento

1. En la instancia de DB2 Database para Linux, UNIX y Windows, emita este mandato para hacer que la instancia compruebe la existencia de valores de tiempo de espera.

   ```
   DB2 UPDATE DBM CFG USING RESYNC_INTERVAL segundos
   ```

   Es necesario que el valor de *segundos* sea menor que el valor mínimo de tiempo de espera que defina para una transacción.
2. En su aplicación, después de crear un objeto XAResource, invoque el método XAResource.setTransactionTimeout para establecer el valor de tiempo de espera.

   Puede comprobar el valor actual de tiempo de espera invocando XAResource.getTransactionTimeout.

Enterprise Java Beans

La arquitectura Enterprise Java Beans es una arquitectura de componentes para el desarrollo y el despliegue de aplicaciones de empresa distribuidas basadas en componentes.

Las aplicaciones que se escriben utilizando la arquitectura Enterprise Java Beans se pueden escribir una sola vez y luego desplegar en cualquier plataforma servidor que sea compatible con la especificación Enterprise Java Beans. Las aplicaciones Java 2 Platform, Enterprise Edition (J2EE) implementan componentes de empresa del lado del servidor mediante Enterprise Java Beans (EJB) que incluyen beans de sesión y beans de entidad.

Los beans de sesión representan servicios de empresa y no se comparten entre usuarios. Los beans de entidad son objetos transaccionales distribuidos, de múltiples usuarios, que representan datos permanentes. Los límites transaccionales de una aplicación EJB se pueden definir especificando transacciones gestionadas por contenedor o gestionadas por bean.

El programa de ejemplo `AccessEmployee.ear` utiliza Enterprise Java Beans para implementar una aplicación J2EE para acceder a una fuente de datos. Encontrará este programa de ejemplo en el directorio `SQLLIB/samples/websphere`.

La aplicación de ejemplo EJB proporciona dos servicios de empresa. Un servicio permite al usuario acceder a información sobre un empleado (que está almacenada en la tabla EMPLOYEE de la base de datos **sample**) mediante el número de empleado de dicho empleado. El otro servicio permite al usuario recuperar una lista de números de empleado de modo que el usuario pueda obtener un número de empleado para utilizarlo para consultar datos del empleado.

El ejemplo siguiente utiliza los EJB para implementar una aplicación J2EE para acceder a una fuente de datos. En el ejemplo se utiliza la arquitectura Model-View-Controller (MVC), que es una arquitectura de GUI de uso habitual. Se utiliza la JSP para implementar la vista (el componente de presentación). Un servlet actúa como controlador en el ejemplo. El servlet controla el flujo de trabajo y delega la petición del usuario al modelo, que se implementa mediante los EJB. El componente modelo del ejemplo consta de dos EJB: un bean de sesión y un bean de entidad. El bean de permanencia gestionada por contenedor (CMP), Employee, representa los objetos transaccionales distribuidos que representan los datos permanentes de la tabla EMPLOYEE de la base de datos sample. El término permanencia gestionada por contenedor significa que el contenedor EJB maneja todo el acceso a base de datos que necesita el bean de entidad. El código del bean no contiene ninguna llamada de acceso a base de datos (SQL). Como resultado, el código del bean no está enlazado a ningún mecanismo de almacenamiento permanente específico (base de datos). El bean de sesión, AccessEmployee, actúa como fachada del bean de entidad y proporciona una estrategia uniforme de acceso de clientes. Este diseño de fachada reduce el tráfico en la red entre el cliente EJB y el bean de entidad y resulta más eficiente en transacciones distribuidas que cuando el cliente EJB accede al bean de entidad directamente. El acceso al servidor de bases de datos se puede proporcionar desde el bean de sesión o el bean de entidad. Los dos servicios de la aplicación de ejemplo muestran ambos métodos para acceder al servidor de bases de datos. En el primer servicio, se utiliza el bean de entidad:

```
//=================================================
// Este método devuelve información sobre un empleado
// mediante la interacción con el bean de entidad
// identificado por el número de empleado proporcionado
public EmployeeInfo getEmployeeInfo(String empNo)
throws java.rmi.RemoteException
}
Employee employee = null;
try
}
employee = employeeHome.findByPrimaryKey(new EmployeeKey(empNo));
EmployeeInfo empInfo = new EmployeeInfo(empNo);
//establecer la información del empleado en el objeto de valor dependiente
empInfo.setEmpno(employee.getEmpno());
empInfo.setFirstName (employee.getFirstName());
empInfo.setMidInit(employee.getMidInit());
empInfo.setLastName(employee.getLastName());
empInfo.setWorkDept(employee.getWorkDept());
empInfo.setPhoneNo(employee.getPhoneNo());
empInfo.setHireDate(employee.getHireDate());
empInfo.setJob(employee.getJob());
empInfo.setEdLevel(employee.getEdLevel());
empInfo.setSex(employee.getSex());
empInfo.setBirthDate(employee.getBirthDate());
empInfo.setSalary(employee.getSalary());
empInfo.setBonus(employee.getBonus());
empInfo.setComm(employee.getComm());
return empInfo;
}
catch (java.rmi.RemoteException rex)
{
......
```

En el segundo servicio, que muestra números de empleado, el bean de sesión, AccessEmployee, accede directamente a la tabla de base de datos.

```
/=============================================
* Obtener lista de números de empleado.
* @return Collection
```

```
*/
public Collection getEmpNoList()
{
ResultSet rs = null;
PreparedStatement ps = null;
Vector list = new Vector();
DataSource ds = null;
Connection con = null;
try
{
ds = getDataSource();
con = ds.getConnection();
String schema = getEnvProps(DBschema);
String query = "Select EMPNO from " + schema + ".EMPLOYEE";
ps = con.prepareStatement(query);
ps.executeQuery();
rs = ps.getResultSet();
EmployeeKey pk;
while (rs.next())
{
pk = new EmployeeKey();
pk.employeeId = rs.getString(1);
list.addElement(pk.employeeId);
}
rs.close();
return list;
```

Capítulo 11. Soporte para la agrupación de conexiones JDBC y SQLJ

La *agrupación de conexiones* forma parte del soporte DataSource de JDBC y está soportada por IBM Data Server Driver para JDBC y SQLJ.

IBM Data Server Driver para JDBC y SQLJ ofrece una fábrica de conexiones agrupadas que utilizan el servidor WebSphere Application Server u otros servidores de aplicaciones. En realidad, el servidor de aplicaciones realiza la agrupación. La agrupación de conexiones es completamente transparente para las aplicaciones JDBC o SQLJ.

La agrupación de conexiones es un sistema para almacenar en antememoria temporalmente conexiones físicas con fuentes de datos, que son equivalentes a hebras de DB2. Cuando JDBC reutiliza conexiones físicas con fuentes de datos, se minimizan las operaciones costosas necesarias para la creación y el cierre posterior de objetos java.sql.Connection.

Cuando no se utiliza la agrupación de conexiones, cada objeto java.sql.Connection representa una conexión física con la fuente de datos. Cuando la aplicación establece una conexión con una fuente de datos, DB2 crea una nueva conexión física con la fuente de datos. Cuando la aplicación invoca el método java.sql.Connection.close, DB2 cierra la conexión física con la fuente de datos.

En cambio, cuando se utiliza la agrupación de conexiones, un objeto java.sql.Connection es una representación lógica y temporal de una conexión física con una fuente de datos. La conexión física con la fuente de datos puede ser reutilizada secuencialmente por instancias de java.sql.Connection. La aplicación puede utilizar el objeto lógico java.sql.Connection exactamente del mismo modo en que utiliza un objeto java.sql.Connection cuando no es posible utilizar la agrupación de conexiones.

Con la agrupación de conexiones, cuando una aplicación JDBC invoca el método DataSource.getConnection, la fuente de datos determinará si existe una conexión física adecuada. Si dicha conexión existe, la fuente de datos devolverá una instancia de java.sql.Connection a la aplicación. Cuando la aplicación JDBC invoca el método java.sql.Connection.close, JDBC no cierre la conexión de la fuente de datos física. En su lugar, JDBC cierra sólo los recursos JDBC, como por ejemplo los objetos Statement o ResultSet. La fuente de datos devuelve la conexión física a la agrupación de conexiones para su reutilización.

Las agrupaciones de conexiones pueden ser *homogéneas* o *heterogéneas*.

Con una agrupación homogénea, todos los objetos Connection que provengan de la misma agrupación de conexiones deben tener las mismas propiedades. El primer objeto Connection lógico que se crea con DataSource tiene las propiedades que se han definido para DataSource. No obstante, una aplicación puede cambiar dichas propiedades. Cuando se devuelve un objeto Connection a la agrupación de conexiones, un servidor de aplicaciones o un módulo de agrupación deberá restaurar los valores originales de las propiedades. Sin embargo, es posible que un servidor de aplicaciones o un módulo de agrupación no pueda restaurar las propiedades modificadas. El controlador JDBC no modifica las propiedades. Por lo

tanto, en función del diseño del servidor de aplicaciones o del módulo de agrupación, es posible que las propiedades de un objeto Connection lógico reutilizado sean las mismas que las que se han definido para DataSource o que sean propiedades diferentes.

Con la agrupación heterogénea, los objetos Connection con propiedades diferentes pueden compartir la misma agrupación de conexiones.

Capítulo 12. Almacenamiento en antememoria de sentencias de IBM Data Server Driver para JDBC y SQLJ

IBM Data Server Driver para JDBC y SQLJ puede utilizar una antememoria de sentencias interna para mejorar el rendimiento de las aplicaciones Java almacenando en antememoria y agrupando las sentencias preparadas.

El almacenamiento interno en antememoria de sentencias está disponible para las conexiones que utilizan IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4, o para las conexiones que utilizan IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS.

El almacenamiento en antememoria interno de sentencias se habilita de cualquiera de las formas siguientes:

- Estableciendo una de las propiedades siguientes en un valor positivo:
  - com.ibm.db2.jcc.DB2ConnectionPoolDataSource.maxStatements, para los objetos que se crean mediante la interfaz javax.sql.ConnectionPoolDataSource.
  - com.ibm.db2.jcc.DB2XADataSource.maxStatements, para los objetos que se crean mediante la interfaz javax.sql.XADataSource.
  - com.ibm.db2.jcc.DB2SimpleDataSource.maxStatements, para los objetos que se crean mediante las interfaces com.ibm.db2.jcc.DB2SimpleDataSource.
- Estableciendo la propiedad maxStatements en un URL y pasando el URL al método DriverManager.getConnection.

Cuando el almacenamiento en antememoria interno de sentencias está habilitado, IBM Data Server Driver para JDBC y SQLJ puede almacenar en antememoria objetos PreparedStatement, objetos CallableStatement y recursos JDBC que utilizan las sentencias SQLJ cuando esos objetos o recursos se cierran de forma lógica. Cuando se invoca explícita o implícitamente el método close en una sentencia, la sentencia se cierra lógicamente.

La reutilización de una sentencia almacenada en antememoria previamente es transparente para las aplicaciones. La antememoria de sentencias existe mientras dure una conexión abierta. Cuando se cierra la conexión, el controlador suprime la antememoria de sentencias y cierra todas las sentencias agrupadas.

Una sentencia abierta lógicamente deja de ser candidata para el almacenamiento en antememoria en cualquiera de las circunstancias siguientes:

- Se produce una excepción en la sentencia.
- Se invoca el método JDBC 4.0 Statement.setPoolable(false).

Cuando IBM Data Server Driver para JDBC y SQLJ intenta almacenar en antememoria una sentencia y la antememoria de sentencias interna está llena, el controlador depura la sentencia almacenada en antememoria que hace más tiempo que se ha utilizado e inserta la nueva sentencia.

La antememoria de sentencias interna se depura en las condiciones siguientes:

- Se emite una sentencia SET que afecta a los objetos de destino de la sentencia de SQL.

- Se ejecuta una sentencia SET que IBM Data Server Driver para JDBC y SQLJ no reconoce.
- IBM Data Server Driver para JDBC y SQLJ detecta que una propiedad que modifica objetos de destino de la sentencia de SQL se ha modificado durante la reutilización de la conexión. currentSchema es un ejemplo de propiedad que modifica los objetos de destino de una sentencia de SQL.

En un programa Java, puede probar si la antememoria de sentencias interna se habilita emitiendo el método DatabaseMetaData.supportsStatementPooling. El método devuelve `true` si se habilita la antememoria de sentencias interna.

IBM Data Server Driver para JDBC y SQLJ no comprueba si las definiciones de los objetos de destino de las sentencias de la antememoria de sentencias interna han cambiado. Si ejecuta sentencias de lenguaje de definición de datos de SQL en una aplicación, tiene que inhabilitar el almacenamiento en antememoria de sentencias interna para esa aplicación.

La antememoria de sentencias interna exige más memoria. Si la memoria queda limitada, puede aumentar el tamaño de la JVM o reducir el valor de maxStatements.

Capítulo 13. Información de consulta sobre JDBC y SQLJ

Las implementaciones de JDBC y SQLJ para IBM proporcionan varias interfaces de programación de aplicaciones, propiedades y mandatos para desarrollar aplicaciones JDBC y SQLJ.

Tipos de datos que se correlacionan con tipos de datos de base de datos en aplicaciones Java

Para escribir programas JDBC y SQLJ que sean efectivos, es necesario que utilice las mejores correlaciones entre tipos de datos Java y tipos de datos de columnas de tabla.

Las tablas siguientes resumen las correlaciones de tipos de datos Java con tipos de datos JDBC y tipos de datos de base de datos para un sistema DB2 Database para Linux, UNIX y Windows, DB2 para z/OS o IBM Informix.

Tipos de datos para actualizar columnas de tabla

La tabla siguiente resume las correlaciones de tipos de datos Java con tipos de datos de base de datos para métodos PreparedStatement.setXXX o ResultSet.updateXXX en programas JDBC, y para expresiones de sistema principal de entrada en programas SQLJ. Cuando aparece listado más de un tipo de datos de Java, el primer tipo de datos es el recomendado.

Tabla 48. Correlaciones de tipos de datos Java con tipos de datos de servidor de bases de datos para actualizar tablas de base de datos

Tipo de datos de Java	Tipo de datos de base de datos
short, java.lang.Short	SMALLINT
boolean[1], byte[1], java.lang.Boolean, java.lang.Byte	SMALLINT
int, java.lang.Integer	INTEGER
long, java.lang.Long	BIGINT[12]
java.math.BigInteger	BIGINT[11]
java.math.BigInteger	CHAR(*n*)[11,5]
float, java.lang.Float	REAL
double, java.lang.Double	DOUBLE
java.math.BigDecimal	DECIMAL(*p*,*s*)[2]
java.math.BigDecimal	DECFLOAT(*n*)[3,4]
java.lang.String	CHAR(*n*)[5]
java.lang.String	GRAPHIC(*m*)[6]
java.lang.String	VARCHAR(*n*)[7]
java.lang.String	VARGRAPHIC(*m*)[8]
java.lang.String	CLOB[9]
java.lang.String	XML[10]
byte[]	CHAR(*n*) FOR BIT DATA[5]
byte[]	VARCHAR(*n*) FOR BIT DATA[7]

Tabla 48. Correlaciones de tipos de datos Java con tipos de datos de servidor de bases de datos para actualizar tablas de base de datos (continuación)

Tipo de datos de Java	Tipo de datos de base de datos
byte[]	BINARY(*n*)[5, 13]
byte[]	VARBINARY(*n*)[7, 13]
byte[]	BLOB[9]
byte[]	ROWID
byte[]	XML[10]
java.sql.Blob	BLOB
java.sql.Blob	XML[10]
java.sql.Clob	CLOB
java.sql.Clob	DBCLOB[9]
java.sql.Clob	XML[10]
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP, TIMESTAMP(*p*), TIMESTAMP WITH TIME ZONE, TIMESTAMP(*p*) WITH TIME ZONE[14,15]
java.io.ByteArrayInputStream	BLOB
java.io.StringReader	CLOB
java.io.ByteArrayInputStream	CLOB
java.io.InputStream	XML[10]
ecom.ibm.db2.jcc.DB2RowID (en desuso)	ROWID
java.sql.RowId	ROWID
com.ibm.db2.jcc.DB2Xml (en desuso)	XML[10]
java.sql.SQLXML	XML[10]
java.util.Date	CHAR(*n*)[11,5]
java.util.Date	VARCHAR(*n*)[11,5]
java.util.Date	DATE[11]
java.util.Date	TIME[11]
java.util.Date	TIMESTAMP, TIMESTAMP(*p*), TIMESTAMP WITH TIME ZONE, TIMESTAMP(*p*) WITH TIME ZONE[11,14,15]
java.util.Calendar	CHAR(*n*)[11,5]
java.util.Calendar	VARCHAR(*n*)[11,5]
java.util.Calendar	DATE[11]
java.util.Calendar	TIME[11]
java.util.Calendar	TIMESTAMP, TIMESTAMP(*p*), TIMESTAMP WITH TIME ZONE, TIMESTAMP(*p*) WITH TIME ZONE[11,14,15]

Tabla 48. Correlaciones de tipos de datos Java con tipos de datos de servidor de bases de datos para actualizar tablas de base de datos (continuación)

Tipo de datos de Java	Tipo de datos de base de datos

Notas:

1. En las actualizaciones de columnas, el servidor de datos no tiene un equivalente exacto para los tipos de datos boolean o byte de Java, pero el más próximo es SMALLINT.
2. *p* es la precisión decimal y *s* es la escala de la columna de tabla.

   Es conveniente que diseñe las aplicaciones financieras de forma que las columnas de tipo java.math.BigDecimal se correlacionen con columnas de tipo DECIMAL. Si conoce la precisión y la escala de una columna DECIMAL, actualizando los datos de la columna DECIMAL con datos de una variable java.math.BigDecimal obtendrá un rendimiento mejor que si utiliza otras combinaciones de tipos de datos.
3. *n*=16 o *n*=34.
4. DECFLOAT es válido para conexiones con servidores de bases de datos DB2 Versión 9.1 para z/OS, DB2 V9.5 para Linux, UNIX y Windows, o DB2 para i V6R1, o bien con servidores de bases de datos posteriores. La utilización de DECFLOAT requiere el SDK de Java versión 5 (1.5) o posterior.
5. *n*<=254.
6. *m*<=127.
7. *n*<=32672.
8. *m*<=16336.
9. Esta correlación solamente es válida si el servidor de bases de datos puede determinar el tipo de datos de la columna.
10. XML es válido para conexiones con servidores de bases de datos DB2 para z/OS Versión 9.1 o posterior, o con servidores de bases de datos DB2 V9.1 para Linux, UNIX y Windows o posterior.
11. Esta correlación sólo es válida para IBM Data Server Driver para JDBC y SQLJ versión 4.13 o posterior.
12. BIGINT es válido para conexiones con servidores de bases de datos DB2 Versión 9.1 para z/OS o posteriores, DB2 V9.1 para Linux, UNIX y Windows o posteriores y todos los servidores de bases de datos DB2 para i soportados.
13. BINARY y VARBINARY son válidos para conexiones con servidores de bases de datos DB2 Versión 9.1 para z/OS o posteriores o con servidores de base de datos DB2 para i5/OS V5R3 y posteriores.
14. *p* indica la precisión de la indicación de fecha y hora, que es el número de dígitos de la parte fraccionaria de la indicación de fecha y hora. 0<=*p*<=12. El valor por omisión es 6. TIMESTAMP(*p*) se soporta en las conexiones con DB2 Database para Linux, UNIX y Windows V9.7 posteriores y DB2 para z/OS V10 y posteriores únicamente.
15. Sólo se admite la cláusula WITH TIME ZONE para conexiones con DB2 para z/OS V10 y posterior.

Tipos de datos para recuperar datos de columnas de tabla

La tabla siguiente resume las correlaciones de tipos de datos DB2 o IBM Informix con tipos de datos Java para métodos ResultSet.getXXX en aplicaciones JDBC, y para iteradores en programas SQLJ. Esta tabla no lista los tipos de objetos de derivador numérico de Java, que se recuperan mediante ResultSet.getObject.

Tabla 49. Correlaciones de tipos de datos de servidor de bases de datos con tipos de datos Java para recuperar datos de tablas de servidor de bases de datos

Tipo de datos de SQL	Tipo de datos de Java o tipo de objeto Java recomendado	Otros tipos de datos Java soportados
SMALLINT	short	byte, int, long, float, double, java.math.BigDecimal, boolean, java.lang.String
INTEGER	int	short, byte, long, float, double, java.math.BigDecimal, boolean, java.lang.String

Tabla 49. Correlaciones de tipos de datos de servidor de bases de datos con tipos de datos Java para recuperar datos de tablas de servidor de bases de datos (continuación)

Tipo de datos de SQL	Tipo de datos de Java o tipo de objeto Java recomendado	Otros tipos de datos Java soportados
BIGINT[5]	long	int, short, byte, float, double, java.math.BigDecimal, boolean, java.lang.String
DECIMAL(*p*,*s*)o NUMERIC(*p*,*s*)	java.math.BigDecimal	long, int, short, byte, float, double, boolean, java.lang.String
DECFLOAT(*n*)[1,2]	java.math.BigDecimal	long, int, short, byte, float, double, java.math.BigDecimal, boolean, java.lang.String
REAL	float	long, int, short, byte, double, java.math.BigDecimal, boolean, java.lang.String
DOUBLE	double	long, int, short, byte, float, java.math.BigDecimal, boolean, java.lang.String
CHAR(*n*)	java.lang.String	long, int, short, byte, float, double, java.math.BigDecimal, boolean, java.sql.Date, java.sql.Time, java.sql.Timestamp, java.io.InputStream, java.io.Reader
VARCHAR(*n*)	java.lang.String	long, int, short, byte, float, double, java.math.BigDecimal, boolean, java.sql.Date, java.sql.Time, java.sql.Timestamp, java.io.InputStream, java.io.Reader
CHAR(*n*) FOR BIT DATA	byte[]	java.lang.String, java.io.InputStream, java.io.Reader
VARCHAR(*n*) FOR BIT DATA	byte[]	java.lang.String, java.io.InputStream, java.io.Reader
BINARY(*n*)[6]	byte[]	Ninguno
VARBINARY(*n*)[6]	byte[]	Ninguno
GRAPHIC(*m*)	java.lang.String	long, int, short, byte, float, double, java.math.BigDecimal, boolean, java.sql.Date, java.sql.Time, java.sql.Timestamp, java.io.InputStream, java.io.Reader
VARGRAPHIC(*m*)	java.lang.String	long, int, short, byte, float, double, java.math.BigDecimal, boolean, java.sql.Date, java.sql.Time, java.sql.Timestamp, java.io.InputStream, java.io.Reader
CLOB(*n*)	java.sql.Clob	java.lang.String
BLOB(*n*)	java.sql.Blob	byte[][3]
DBCLOB(*m*)	No existe un equivalente exacto. Utilice java.sql.Clob.	
ROWID	java.sql.RowId	byte[], com.ibm.db2.jcc.DB2RowID (en desuso)
XML[4]	java.sql.SQLXML	byte[], java.lang.String, java.io.InputStream, java.io.Reader

Tabla 49. Correlaciones de tipos de datos de servidor de bases de datos con tipos de datos Java para recuperar datos de tablas de servidor de bases de datos (continuación)

Tipo de datos de SQL	Tipo de datos de Java o tipo de objeto Java recomendado	Otros tipos de datos Java soportados
DATE	java.sql.Date	java.sql.String, java.sql.Timestamp
TIME	java.sql.Time	java.sql.String, java.sql.Timestamp
TIMESTAMP, TIMESTAMP(*p*), TIMESTAMP WITH TIME ZONE, TIMESTAMP(*p*) WITH TIME ZONE[7,8]	java.sql.Timestamp	java.sql.String, java.sql.Date, java.sql.Time, java.sql.Timestamp

Notas:

1. *n*=16 o *n*=34.
2. DECFLOAT es válido para conexiones con servidores de bases de datos DB2 Versión 9.1 para z/OS, DB2 V9.5 para Linux, UNIX y Windows, o DB2 para i V6R1, o bien con servidores de bases de datos posteriores. La utilización de DECFLOAT requiere el SDK de Java versión 5 (1.5) o posterior.
3. Esta correlación solamente es válida si el servidor de bases de datos puede determinar el tipo de datos de la columna.
4. XML es válido para conexiones con servidores de bases de datos DB2 para z/OS Versión 9.1 o posterior, o con servidores de bases de datos DB2 V9.1 para Linux, UNIX y Windows o posterior.
5. BIGINT es válido para conexiones con servidores de bases de datos DB2 Versión 9.1 para z/OS o posteriores, DB2 V9.1 para Linux, UNIX y Windows o posteriores y todos los servidores de bases de datos DB2 para i soportados.
6. BINARY y VARBINARY son válidos para conexiones con servidores de bases de datos DB2 Versión 9.1 para z/OS o posteriores o con servidores de base de datos DB2 para i5/OS V5R3 o posteriores.
7. *p* indica la precisión de la indicación de fecha y hora, que es el número de dígitos de la parte fraccionaria de la indicación de fecha y hora. 0<=*p*<=12. El valor por omisión es 6. TIMESTAMP(*p*) se soporta en las conexiones con DB2 Database para Linux, UNIX y Windows V9.7 posteriores y DB2 para z/OS V10 y posteriores únicamente.
8. Sólo se admite la cláusula WITH TIME ZONE para conexiones con DB2 para z/OS V10 y posterior.

Tipos de datos para invocar procedimientos almacenados y funciones definidas por el usuario

La tabla siguiente resume las correlaciones de tipos de datos Java con tipos de datos JDBC y tipos de datos DB2 o IBM Informix para invocar parámetros de funciones definidas por el usuario y procedimientos almacenados. Las correlaciones de tipos de datos Java con tipos de datos JDBC son para métodos CallableStatement.registerOutParameter en programas JDBC. Las correlaciones de tipos de datos Java con tipos de datos de servidor de bases de datos son para parámetros utilizados al invocar procedimientos almacenados o funciones definidas por el usuario.

Si la tabla siguiente lista más de un tipo de datos Java, el primer tipo de datos es el tipo de datos **recomendado**.

Tabla 50. Correlaciones de tipos de datos Java, JDBC y SQL para invocar procedimientos almacenados y funciones definidas por el usuario

Tipo de datos de Java	Tipo de datos de JDBC	Tipo de datos de SQL
boolean[3], java.lang.Boolean	BOOLEAN	BOOLEAN[1,2]
boolean[3], java.lang.Boolean	BIT	SMALLINT
byte[3], java.lang.Byte	TINYINT	SMALLINT
short, java.lang.Short	SMALLINT	SMALLINT
int, java.lang.Integer	INTEGER	INTEGER
long, java.lang.Long	BIGINT	BIGINT[7]

Tabla 50. Correlaciones de tipos de datos Java, JDBC y SQL para invocar procedimientos almacenados y funciones definidas por el usuario (continuación)

Tipo de datos de Java	Tipo de datos de JDBC	Tipo de datos de SQL
float, java.lang.Float	REAL	REAL
float, java.lang.Float	FLOAT	REAL
double, java.lang.Double	DOUBLE	DOUBLE
java.math.BigDecimal	NUMERIC	DECIMAL
java.math.BigDecimal	DECIMAL	DECIMAL
java.math.BigDecimal	java.types.OTHER	DECFLOAT*n*[4]
java.math.BigDecimal	com.ibm.db2.jcc.DB2Types.DECFLOAT	DECFLOAT*n*[4]
java.lang.String	CHAR	CHAR
java.lang.String	CHAR	GRAPHIC
java.lang.String	VARCHAR	VARCHAR
java.lang.String	VARCHAR	VARGRAPHIC
java.lang.String	LONGVARCHAR	VARCHAR
java.lang.String	VARCHAR	CLOB
java.lang.String	LONGVARCHAR	CLOB
java.lang.String	CLOB	CLOB
byte[]	BINARY	CHAR FOR BIT DATA
byte[]	VARBINARY	VARCHAR FOR BIT DATA
byte[]	BINARY	BINARY[6]
byte[]	VARBINARY	VARBINARY[6]
byte[]	LONGVARBINARY	VARCHAR FOR BIT DATA
byte[]	VARBINARY	BLOB[5]
byte[]	LONGVARBINARY	BLOB[5]
java.sql.Date	DATE	DATE
java.sql.Time	TIME	TIME
java.sql.Timestamp	TIMESTAMP	TIMESTAMP, TIMESTAMP(*p*), TIMESTAMP WITH TIME ZONE, TIMESTAMP(*p*) WITH TIME ZONE[8,9]
java.sql.Blob	BLOB	BLOB
java.sql.Clob	CLOB	CLOB
java.sql.Clob	CLOB	DBCLOB
java.io.ByteArrayInputStream	Ninguno	BLOB
java.io.StringReader	Ninguno	CLOB
java.io.ByteArrayInputStream	Ninguno	CLOB
ecom.ibm.db2.jcc.DB2RowID (en desuso)	com.ibm.db2.jcc.DB2Types.ROWID	ROWID
java.sql.RowId	java.sql.Types.ROWID	ROWID
com.ibm.db2.jcc.DB2Xml (en desuso)	com.ibm.db2.jcc.DB2Types.XML	XML AS CLOB

Tabla 50. Correlaciones de tipos de datos Java, JDBC y SQL para invocar procedimientos almacenados y funciones definidas por el usuario (continuación)

Tipo de datos de Java	Tipo de datos de JDBC	Tipo de datos de SQL
java.sql.SQLXML	java.sql.Types.SQLXML	XML
java.sql.SQLXML	java.sql.Types.SQLXML	XML AS CLOB
java.sql.Array	java.sql.Types.ARRAY	ARRAY[2]
java.sql.Struct	java.sql.Types.STRUCT	ROW[1,2]
java.sql.ResultSet	com.ibm.db2.jcc.DB2Types.CURSOR	tipo CURSOR[2]

Notas:

1. Este tipo de datos de parámetros únicamente se soporta en IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 para DB2 Database para Linux, UNIX y Windows.
2. Este tipo de datos de parámetros únicamente se soporta para las llamadas de procedimiento almacenado.
3. Un procedimiento almacenado o función definida por el usuario que esté definido con un parámetro de tipo SMALLINT se puede invocar con un parámetro de tipo boolean o byte. Sin embargo, esto no es aconsejable.
4. Los parámetros DECFLOAT de las rutinas Java solamente son válidos para conexiones con servidores de bases de datos DB2 para z/OS versión 9.1 o posterior. Los parámetros DECFLOAT en rutinas Java no se admiten para conexiones con para Linux, UNIX y Windows o DB2 para i. La utilización de DECFLOAT requiere el SDK de Java Versión 5 (1.5) o posterior.
5. Esta correlación solamente es válida si el servidor de bases de datos puede determinar el tipo de datos de la columna.
6. BINARY y VARBINARY son válidos para conexiones con servidores de bases de datos DB2 Versión 9.1 para z/OS o posteriores o con servidores de base de datos DB2 para i5/OS V5R3 y posteriores.
7. BIGINT es válido para conexiones con servidores de bases de datos DB2 Versión 9.1 para z/OS o posteriores, DB2 V9.1 para Linux, UNIX y Windows o posteriores y todos los servidores de bases de datos DB2 para i soportados.
8. *p* indica la precisión de la indicación de fecha y hora, que es el número de dígitos de la parte fraccionaria de la indicación de fecha y hora. 0<=*p*<=12. El valor por omisión es 6. TIMESTAMP(*p*) se soporta en las conexiones con DB2 Database para Linux, UNIX y Windows V9.7 posteriores y DB2 para z/OS V10 y posteriores únicamente.
9. Sólo se admite la cláusula WITH TIME ZONE para conexiones con DB2 para z/OS V10 y posterior.

Tipos de datos en procedimientos almacenados Java y funciones definidas por el usuario

La tabla siguiente resume las correlaciones de tipos de datos de parámetros de SQL contenidos en una sentencia CREATE PROCEDURE o CREATE FUNCTION con los tipos de datos contenidos en el correspondiente método de procedimiento almacenado Java o función definida por el usuario.

Para DB2 Database para Linux, UNIX y Windows, si la lista incluye más de un tipo de datos Java para un tipo de datos SQL, solamente es válido el **primer** tipo de datos Java.

Para DB2 para z/OS, si se lista más de un tipo de datos de Java y se utiliza un tipo de datos distinto del primero como parámetro de un método, es necesario incluir una signatura de método en la cláusula EXTERNAL de la sentencia CREATE PROCEDURE o CREATE FUNCTION que especifique los tipos de datos Java de los parámetros del método.

Tabla 51. Correlaciones de los tipos de datos de SQL de las sentencias CREATE PROCEDURE y CREATE FUNCTION con los tipos de datos del programa correspondiente de función definida por el usuario o procedimiento almacenado de Java

Tipo de datos de SQL en CREATE PROCEDURE o CREATE FUNCTION	Tipo de datos en procedimiento almacenado Java o función definida por el usuario[2]
SMALLINT	short, java.lang.Integer
BOOLEAN[1]	boolean
INTEGER	int, java.lang.Integer
BIGINT[3]	long, java.lang.Long
REAL	float, java.lang.Float
DOUBLE	double, java.lang.Double
DECIMAL	java.math.BigDecimal
DECFLOAT[4]	java.math.BigDecimal
CHAR	java.lang.String
VARCHAR	java.lang.String
CHAR FOR BIT DATA	byte[]
VARCHAR FOR BIT DATA	byte[]
BINARY[5]	byte[]
VARBINARY[5]	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP, TIMESTAMP(*p*), TIMESTAMP WITH TIME ZONE, TIMESTAMP(*p*) WITH TIME ZONE[6,7]	java.sql.Timestamp
BLOB	java.sql.Blob
CLOB	java.sql.Clob
DBCLOB	java.sql.Clob
ROWID	java.sql.Types.ROWID
ARRAY[1]	java.sql.Array
ROW[1]	java.sql.Struct
XML AS CLOB	java.sql.Types.SQLXML

Notas:

1. Este tipo de datos de parámetros únicamente se soporta para los procedimientos almacenados.
2. En el caso de un procedimiento almacenado o una función definida por el usuario en un servidor DB2 Database para Linux, UNIX y Windows, sólo es válido el **primer** tipo de datos.
3. BIGINT es válido para conexiones con servidores de bases de datos DB2 para z/OS Versión 9.1 o posterior, o con servidores de bases de datos DB2 V9.1 para Linux, UNIX y Windows o posterior.
4. Los parámetros DECFLOAT de las rutinas Java solamente son válidos para conexiones con servidores de bases de datos DB2 para z/OS versión 9.1 o posterior. Los parámetros DECFLOAT en rutinas Java no se admiten para conexiones con para Linux, UNIX y Windows o DB2 para i. La utilización de DECFLOAT requiere el SDK de Java Versión 5 (1.5) o posterior.
5. BINARY y VARBINARY son válidos para conexiones con servidores de bases de datos DB2 Versión 9.1 para z/OS o posteriores.
6. *p* indica la precisión de la indicación de fecha y hora, que es el número de dígitos de la parte fraccionaria de la indicación de fecha y hora. 0<=*p*<=12. El valor por omisión es 6. TIMESTAMP(*p*) se soporta en las conexiones con DB2 Database para Linux, UNIX y Windows V9.7 posteriores y DB2 para z/OS V10 y posteriores únicamente.
7. Sólo se admite la cláusula WITH TIME ZONE para conexiones con DB2 para z/OS V10 y posterior.

Valores de fecha, de hora, y de indicación de fecha y hora que pueden causar problemas en aplicaciones JDBC y SQLJ

Puede obtener resultados inesperados en aplicaciones JDBC y SQLJ si utiliza valores de fecha, de hora, y de indicación de fecha y hora que no corresponden a fechas y horas reales.

Los elementos siguientes pueden generar problemas:

- Uso de la hora '24' para representar medianoche
- Uso de una fecha entre el 5 de octubre de 1582 y el 14 de octubre de 1582, ambos inclusive

Problemas con el uso de la hora '24' para representar medianoche

IBM Data Server Driver para JDBC y SQLJ utiliza tipos de datos Java para el proceso interno de parámetros de entrada y salida y el contenido de ResultSet en las aplicaciones JDBC y SQLJ. El controlador utiliza el tipo de datos Java que mejor se adecue al correspondiente tipo de datos SQL cuando el tipo de datos SQL de destino es conocido por el controlador.

Para los valores que se asignan a los tipos DATE, TIME o TIMESTAMP de SQL o que se recuperan de ellos, IBM Data Server Driver para JDBC y SQLJ utiliza java.sql.Date para los tipos DATE de SQL, java.sql.Time para los tipos TIME de SQL, y java.sql.Timestamp para los tipos TIMESTAMP de SQL.

Cuando asigna un valor de tipo DATE, TIME, o TIMESTAMP, IBM Data Server Driver para JDBC y SQLJ utiliza recursos de Java para convertir el valor a un valor de tipo java.sql.Date, java.sql.Time, o java.sql.Timestamp. Si la representación de tipo carácter de un valor de fecha, de hora o de indicación de fecha y hora no corresponde a una fecha ni hora real, Java ajusta el valor para convertirlo en un valor de fecha u hora real. En particular, Java convierte el valor de hora '24' en la hora '00' del día siguiente. Este ajuste puede producir una excepción para un valor de indicación de fecha y hora igual a '9999-12-31 24:00:00.0', pues el valor de año ajustado pasa a ser '10000'.

Importante: Para evitar resultados inesperados cuando asigna o recupera valores de fecha, de hora, o de indicación de fecha y hora en aplicaciones JDBC o SQLJ, compruebe que los valores sean reales. Además, no utilice '24' como componente representativo de las horas en un valor de hora, o de fecha y hora.

Si un valor que no corresponde a una fecha u hora real, tal como un valor cuyo componente representativo de las horas es '24', se almacena en una columna TIME o TIMESTAMP, puede evitar el ajuste de valor que se realiza durante la obtención del valor ejecutando la función CHAR de SQL para esa columna en la sentencia SELECT por la que se define un conjunto de resultados (ResultSet). La función CHAR convierte el valor de fecha u hora en un valor de tipo carácter en la base de datos. Pero si utiliza el método getTime o getTimestamp para recuperar ese valor a partir de ResultSet, IBM Data Server Driver para JDBC y SQLJ convierte el valor en un tipo java.sql.Time o java.sql.Timestamp, y Java ajusta el valor. Para evitar el ajuste de la fecha, ejecute la función CHAR para el valor de la columna, *y* recupere el valor de ResultSet mediante el método getString.

Los ejemplos siguientes muestran los resultados de actualizar columnas DATE, TIME o TIMESTAMP en aplicaciones JDBC o SQLJ, cuando los datos de la aplicación no representan fechas ni horas reales,

Tabla 52. Ejemplos de actualización de valores DATE, TIME o TIMESTAMP de SQL mediante valores de fecha, hora, o fecha y hora de Java que no representan fechas ni horas reales.

Valor de entrada de tipo serie	Tipo de destino en la base de datos	Valor enviado a columna de tabla, o excepción
2008-13-35	DATE	2009-02-04
25:00:00	TIME	01:00:00
24:00:00	TIME	00:00:00
2008-15-36 28:63:74.0	TIMESTAMP	2009-04-06 05:04:14.0
9999-12-31 24:00:00.0	TIMESTAMP	Excepción, pues el valor ajustado (10000-01-01 00:00:00.0) es mayor que el año máximo 9999.

Los ejemplos siguientes muestran los resultados de recuperar datos a partir de columnas TIMESTAMP en aplicaciones JDBC o SQLJ, cuando los valores de esas columnas no representan fechas ni horas reales,

Tabla 53. Resultados de recuperar valores DATE, TIME o TIMESTAMP de SQL en variables de aplicación Java cuando los valores no representan fechas ni horas reales

Sentencia SELECT	Valor en la columna TS_COL de tipo TIMESTAMP	Tipo de destino en la aplicación (método getXXX de recuperación)	Valor recuperado a partir de la columna de tabla
SELECT TS_COL FROM TABLE1	2000-01-01 24:00:00.000000	java.sql.Timestamp (getTimestamp)	2000-01-02 00:00:00.000000
SELECT TS_COL FROM TABLE1	2000-01-01 24:00:00.000000	String (getString)	2000-01-02 00:00:00.000000
SELECT CHAR(TS_COL) FROM TABLE1	2000-01-01 24:00:00.000000	java.sql.Timestamp (getTimestamp)	2000-01-02 00:00:00.000000
SELECT CHAR(TS_COL) FROM TABLE1	2000-01-01 24:00:00.000000	String (getString)	2000-01-01 24:00:00.000000 (Java no realiza ningún ajuste)

Problemas con el uso de fechas en el rango entre el 5 de octubre de 1582 y el 14 de octubre de 1582

Las clases Java java.util.Date y java.util.Timestamp utilizan el calendario juliano para fechas anteriores al 4 de octubre de 1582 y el calendario gregoriano para fechas que comienzan el 4 de octubre de 1582. En el calendario gregoriano, el 4 de octubre de 1582 viene seguido por el 15 de octubre de 1582. Si un programa Java detecta un valor java.util.Date o java.util.Timestamp que se halla entre el 5 de octubre de 1582 y el 14 de octubre de 1582, ambos inclusive, Java añade 10 días a esa fecha. Por esta razón, un valor DATE o TIMESTAMP en una tabla de DB2 que tenga un valor entre el 5 y el 14 de octubre de 1582, ambos inclusive, se recupera en un programa Java como un valor java.util.Date o java.util.Timestamp entre el 15 y el 24 de octubre de 1582, ambos inclusive. Un valor java.util.Date o java.util.Timestamp en un programa Java que esté incluido entre el 5 y el 14 de octubre de 1582 se almacena en una tabla de DB2 como un valor DATE o TIMESTAMP entre el 15 y el 24 de octubre de 1582, ambos inclusive.

Ejemplo: Recuperar la fecha 10 de octubre de 1582, desde una columna DATE.

```
// DATETABLE has one date column with one row.
// Its value is 1582-10-10.
java.sql.ResultSet rs =
 statement.executeQuery(select * from DATETABLE);
rs.next();
System.out.println(rs.getDate(1)); // Value is retrieved as 1582-10-20
```

Ejemplo: Almacenar la fecha 10 de octubre de 1582, en una columna DATE.

```
java.sql.Date d = java.sql.Date.valueOf("1582-10-10");
java.sql.PreparedStatement ps =
 c.prepareStatement("Insert into DATETABLE values(?)");
ps.setDate(1, d);
ps.executeUpdate(); // Value is inserted as 1582-10-20
```

Para recuperar un valor situado entre el 5 y el 14 de octubre de 1582, desde una tabla de DB2 sin ningún ajuste de fecha, ejecute la función SQL CHAR contra la columna DATE o TIMESTAMP en la sentencia SELECT que define un ResultSet. La función CHAR convierte el valor de fecha u hora en un valor de tipo carácter en la base de datos.

Para almacenar un valor situado entre el 5 y el 14 de octubre de 1582 en una tabla de DB2 sin ningún ajuste de fecha, se puede usar una de las técnicas siguientes:

- Para una aplicación JDBC o SQLJ, utilice el método setString para asignar el valor a un parámetro de entrada de tipo String. Convierta el parámetro de entrada en VARCHAR y ejecute la función DATE o TIMESTAMP contra el resultado de la conversión. A continuación, almacene el resultado de la función DATE o TIMESTAMP en la columna DATE o TIMESTAMP.
- Para una aplicación JDBC, establezca la propiedad sendDataAsIs de Connection o DataSource en **true** y utilice el método setString para asignar el valor de fecha o de indicación de fecha y hora al parámetro de entrada. A continuación, ejecute una sentencia de SQL para asignar el valor de tipo String en la columna DATE o TIMESTAMP.

Ejemplo: Recuperar la fecha 10 de octubre de 1582, desde una columna DATE sin realizar ningún ajuste de fecha.

```
// DATETABLE has one date column called DATECOL with one row.
// Its value is 1582-10-10.
java.sql.ResultSet rs =
 statement.executeQuery(SELECT CHAR(DATECOL) FROM DATETABLE);
rs.next();
System.out.println(rs.getString(1)); // Value is retrieved as 1582-10-10
```

Ejemplo: Almacenar la fecha 10 de octubre de 1582, en una columna DATE sin realizar ningún ajuste de fecha.

```
String s = "1582-10-10";
java.sql.Statement stmt = c.createStatement;
java.sql.PreparedStatement ps =
 c.prepareStatement("Insert INTO DATETABLE VALUES " +
  "(DATE(CAST (? AS VARCHAR)))");
ps.setString(1, s);
ps.executeUpdate(); // Value is inserted as 1582-10-10
```

Pérdida de datos de indicación de fecha y hora en aplicaciones JDBC y SQLJ

Para DB2 para z/OS Versión 10 o posteriores, o DB2 Database para Linux, UNIX y Windows Versión 9.7 o posteriores, puede especificarse la precisión de la parte fraccionaria de una columna TIMESTAMP, con una precisión máxima de 12 dígitos. La parte fraccionaria de un valor de indicación de fecha y hora de Java puede

tener hasta 9 dígitos de precisión. Dependiendo de la definición de columna, pueden perderse datos al actualizar una columna TIMESTAMP(*p*) o al recuperar datos de una columna TIMESTAMP(*p*).

Pérdida de datos para los datos de entrada

Si utiliza una llamada setTimestamp para pasar un valor de indicación de fecha y hora a una columna TIMESTAMP(*p*), la precisión máxima del valor de Java que se envía a la fuente de datos es de 9. Si utiliza una llamada setTimestamp para pasar un valor de indicación de fecha y hora a una columna TIMESTAMP en una fuente de datos que no admite TIMESTAMP(*p*), la precisión máxima del valor de Java que se envía a la fuente de datos es de 6. En lo que respecta a la entrada a una columna TIMESTAMP(*p*), si la precisión de la columna de destino es inferior a la precisión del valor de entrada, la fuente de datos trunca los dígitos sobrantes en la parte fraccionaria de la indicación de fecha y hora.

Si utiliza una llamada setString para pasar el valor de entrada, puede enviar un valor con una precisión mayor que 9 a la fuente de datos.

Para IBM Data Server Driver para JDBC y SQLJ versión 3.59 o posterior, no se produce pérdida de datos si la columna TIMESTAMP(*p*) es lo suficientemente grande como para acomodar el valor de entrada. Para IBM Data Server Driver para JDBC y SQLJ versión 3.58 o posterior, la pérdida de datos depende de la configuración de las propiedades deferPrepares y sendDataAsIs:

- Si sendDataAsIs se establece en `true`, IBM Data Server Driver para JDBC y SQLJ envía la serie a la fuente de datos sin modificarla, de forma que la parte fraccionaria del valor de indicación de fecha y hora puede superar los 9 dígitos. Si el valor de *p* en la columna TIMESTAMP(*p*) es igual o mayor que el número de dígitos de la parte fraccionaria de los datos de entrada, no se produce pérdida de datos.
- Si sendDataAsIs se establece en `false`, la pérdida de datos depende de la configuración de deferPrepares.
- Si deferPrepares se establece en `true`, en la *primera* ejecución de una sentencia UPDATE, IBM Data Server Driver para JDBC y SQLJ envía la serie a la fuente de datos sin modificarla, de forma que la parte fraccionaria del valor de indicación de fecha y hora puede superar los 9 dígitos. Si el valor de *p* en la columna TIMESTAMP(*p*) es igual o mayor que el número de dígitos de la parte fraccionaria de los datos de entrada, no se produce pérdida de datos.

  Para las ejecuciones posteriores de la sentencia UPDATE, IBM Data Server Driver para JDBC y SQLJ puede determinar que el tipo de datos de destino es un tipo de datos TIMESTAMP. Si la fuente de datos admite columnas TIMESTAMP(*p*), el controlador convierte el valor de entrada a un valor java.sql.Timestamp con una precisión de 9 como máximo. Si la fuente de datos no admite columnas TIMESTAMP(*p*), el controlador convierte el valor de entrada a un valor java.sql.Timestamp con una precisión de 6 como máximo. Se produce pérdida de datos si la precisión del valor original es mayor que la precisión del valor java.sql.Timestamp convertido, o si la precisión del valor java.sql.Timestamp es mayor que la precisión de la columna TIMESTAMP(*p*).
- Si deferPrepares se establece en `false`, IBM Data Server Driver para JDBC y SQLJ puede determinar que el tipo de datos de destino es un tipo de datos TIMESTAMP. Si la fuente de datos admite columnas TIMESTAMP(*p*), el controlador convierte el valor de entrada a un valor java.sql.Timestamp con una precisión de 9 como máximo. Si la fuente de datos no admite columnas TIMESTAMP(*p*), el controlador convierte el valor de entrada a un valor java.sql.Timestamp con una precisión de 6 como máximo. Se produce pérdida de

datos si la precisión del valor original es mayor que la precisión del valor java.sql.Timestamp convertido, o si la precisión del valor java.sql.Timestamp es mayor que la precisión de la columna TIMESTAMP(*p*).

Puede reducir la pérdida de datos en los valores de indicación de fecha y hora de entrada utilizando una llamada setString y estableciendo sendDataAsIs en `true`. No obstante, si establece sendDataAsIs en `true`, tendrá que asegurarse de que los tipos de datos de la aplicación sean compatibles con los tipos de datos de la fuente de datos.

Pérdida de datos para los datos de salida

Cuando utiliza una llamada getTimestamp o getString para recuperar datos de una columna TIMESTAMP(*p*), IBM Data Server Driver para JDBC y SQLJ convierte el valor a un valor java.sql.Timestamp con una precisión de 9 como máximo. Si la precisión del valor fuente es mayor que 9, el controlador trunca la parte fraccionaria del valor recuperado en nueve dígitos. Si no desea que se produzca truncamiento, en la sentencia SELECT que recupera el valor TIMESTAMP(*p*), puede convertir el valor TIMESTAMP(*p*) en un tipo de datos de caracteres, como VARCHAR, y utilizar getString para recuperar el valor de ResultSet.

Recuperación de valores especiales de columnas DECFLOAT en aplicaciones Java

Es necesario aplicar un tratamiento especial si recupera valores de las columnas DECFLOAT y las columnas DECFLOAT contienen los valores NaN, Infinity o -Infinity.

El tipo de datos Java recomendado para la recuperación de valores de columnas DECFLOAT es java.math.BigDecimal. Sin embargo, recibe un código de error de SQL -4231 si realiza cualquiera de estas operaciones:

- Recuperar el valor NaN, Infinity o -Infinity de una columna DECFLOAT utilizando el método JDBC java.sql.ResultSet.getBigDecimal o java.sql.ResultSet.getObject
- Recuperar el valor NaN, Infinity o -Infinity de una columna DECFLOAT en una variable java.math.BigDecimal en una cláusula SQLJ de un programa SQLJ

Podrá eludir esta restricción realizando pruebas para el error -4231 y recuperando el valor especial mediante la utilización del método java.sql.ResultSet.getDouble o java.sql.ResultSet.getString.

Supongamos que se utilizaron las sentencias de SQL siguientes para crear y llenar una tabla.

```
CREATE TABLE TEST.DECFLOAT_TEST
(
 INT_VAL INT,
 DECFLOAT_VAL DECFLOAT
);
INSERT INTO TEST.DECFLOAT_TEST (INT_VAL, DECFLOAT_VAL) VALUES (1, 123.456),
INSERT INTO TEST.DECFLOAT_TEST (INT_VAL, DECFLOAT_VAL) VALUES (2, INFINITY),
INSERT INTO TEST.DECFLOAT_TEST (INT_VAL, DECFLOAT_VAL) VALUES (3, -123.456),
INSERT INTO TEST.DECFLOAT_TEST (INT_VAL, DECFLOAT_VAL) VALUES (4, -INFINITY),
INSERT INTO TEST.DECFLOAT_TEST (INT_VAL, DECFLOAT_VAL) VALUES (5, NaN);
```

El código siguiente recupera el contenido de la columna DECFLOAT utilizando el método java.sql.ResultSet.getBigDecimal. Si falla la recuperación porque el valor de

columna es NaN, INFINITY o -INFINITY, el programa recupera el valor utilizando el método java.sql.ResultSet.getBigDouble.

```
final static int DECFLOAT_SPECIALVALUE_ENCOUNTERED = -4231;
java.sql.Connection con =
  java.sql.DriverManager.getConnection("jdbc:db2://localhost:50000/sample"
    , "userid", "password");
java.sql.Statement stmt = con.createStatement();
java.sql.ResultSet rs = stmt.executeQuery(
 "SELECT INT_VAL, DECFLOAT_VAL FROM TEST.DECFLOAT_TEST ORDER BY INT_VAL");
int i = 0;
while (rs.next()) {
 try {
  System.out.println("\nRow " + ++i);
  System.out.println("INT_VAL      = " + rs.getInt(1));
  System.out.println("DECFLOAT_VAL = " + rs.getBigDecimal(2));
 }
 catch(java.sql.SQLException e) {
  System.out.println("Caught SQLException" + e.getMessage());
  if (e.getErrorCode() == DECFLOAT_SPECIALVALUE_ENCOUNTERED) {
  // getBigDecimal ha fallado porque el valor recuperado es NaN,
  // INFINITY o -INFINITY; volver a intentar con getDouble.
     double d = rs.getDouble(2);
     if (d == Double.POSITIVE_INFINITY) {
       System.out.println("DECFLOAT_VAL = +INFINITY");
     } else if (d == Double.NEGATIVE_INFINITY) {
       System.out.println("DECFLOAT_VAL = -INFINITY");
     } else if (d == Double.NaN) {
       System.out.println("DECFLOAT_VAL = NaN");
     } else {
       System.out.println("DECFLOAT_VAL = " + d);
     }
  } else {
     e.printStackTrace();
    }
 }
```

El código siguiente recupera el contenido de la columna DECFLOAT utilizando el método java.sql.ResultSet.getBigDecimal. Si falla la recuperación porque el valor de columna es NaN, INFINITY o -INFINITY, el programa recupera el valor utilizando el método java.sql.ResultSet.getString.

```
final static int DECFLOAT_SPECIALVALUE_ENCOUNTERED = -4231;
java.sql.Connection con =
  java.sql.DriverManager.getConnection("jdbc:db2://localhost:50000/sample"
    , "userid", "password");
java.sql.Statement stmt = con.createStatement();
java.sql.ResultSet rs = stmt.executeQuery(
 "SELECT INT_VAL, DECFLOAT_VAL FROM TEST.DECFLOAT_TEST ORDER BY INT_VAL");
int i = 0;
while (rs.next()) {
 try {
  System.out.println("\nRow " + ++i);
  System.out.println("INT_VAL      = " + rs.getInt(1));
  System.out.println("DECFLOAT_VAL = " + rs.getBigDecimal(2));
 }
 catch(java.sql.SQLException e) {
  System.out.println("Caught SQLException" + e.getMessage());
  if (e.getErrorCode() == DECFLOAT_SPECIALVALUE_ENCOUNTERED) {
  // getBigDecimal ha fallado porque el valor recuperado es NaN,
  // INFINITY o -INFINITY; volver a intentar con getString.
     System.out.println("DECFLOAT_VAL = "+rs.getString(2));
  } else {
     e.printStackTrace();
    }
 }
```

Propiedades de IBM Data Server Driver para JDBC y SQLJ

Las propiedades de IBM Data Server Driver para JDBC y SQLJ definen cómo se debe crear la conexión con una fuente de datos determinada. La mayoría de las propiedades se pueden definir para un objeto DataSource o para un objeto Connection.

Métodos para establecer las propiedades

Puede definir propiedades de una de las maneras siguientes:

- Utilizando métodos setXXX, donde `XXX` es el nombre no calificado de la propiedad, con el primer carácter en mayúsculas.

  Las propiedades son aplicables a las siguientes implementaciones específicas de IBM Data Server Driver para JDBC y SQLJ que heredan las propiedades de com.ibm.db2.jcc.DB2BaseDataSource:
  - com.ibm.db2.jcc.DB2SimpleDataSource
  - com.ibm.db2.jcc.DB2ConnectionPoolDataSource
  - com.ibm.db2.jcc.DB2XADataSource
- En un valor java.util.Properties del parámetro *info* de una llamada DriverManager.getConnection.
- En un valor `java.lang.String` del parámetro *url* de una llamada DriverManager.getConnection.

  Algunas propiedades con un tipo de datos int tienen valores de campo constante predefinidos. Hay que resolver los valores de campo constante predefinidos con sus valores enteros antes de utilizar dichos valores en el parámetro *url*. Por ejemplo, no se puede utilizar com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL en un parámetro *url*. Sin embargo, se puede crear una serie URL que incluya com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL, y asignar la serie URL a una variable String. A continuación, se puede utilizar la variable String en el parámetro *url*:

```
String url =
 "jdbc:db2://sysmvs1.stl.ibm.com:5021/STLEC1" +
 ":user=dbadm;password=dbadm;" +
 "traceLevel=" +
 (com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL) + ";";

 Connection con =
   java.sql.DriverManager.getConnection(url);
```

Propiedades comunes de IBM Data Server Driver para JDBC y SQLJ para todos los productos de base de datos permitidos

La mayoría de las propiedades de IBM Data Server Driver para JDBC y SQLJ son aplicables a todos los productos de base de datos que son compatibles con el controlador.

A menos que se indique otra cosa, todas las propiedades están contenidas en com.ibm.db2.jcc.DB2BaseDataSource.

Estas propiedades son las siguientes:

`affinityFailbackInterval`
Especifica la longitud del intervalo, en segundos, que espera IBM Data Server Driver para JDBC y SQLJ antes de intentar transferir una conexión existente al servidor primario. Un valor igual o menor que 0 significa que la conexión no se transfiere. El valor por omisión es `DB2BaseDataSource.NOT_SET` (0).

Los intentos de restablecer las conexiones con el servidor primario se realizan en los límites de transacción, después de haber transcurrido el intervalo especificado.

affinityFailbackInterval se utiliza únicamente si los valores de las propiedades enableSeamlessFailover y enableClientAffinitiesList son `DB2BaseDataSource.YES` (1).

affinityFailbackInterval se aplica únicamente a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4.

allowNextOnExhaustedResultSet
Especifica cómo IBM Data Server Driver para JDBC y SQLJ maneja una llamada de ResultSet.next() a un cursor de solo avance que está posicionado después de la última fila de ResultSet. El tipo de datos de esta propiedad es int.

Los valores posibles son:

DB2BaseDataSource.YES (1)
Para un ResultSet que se ha definido como TYPE_FORWARD_ONLY, ResultSet.next() devuelve `false` (falso) si el cursor anteriormente se ha posicionado después de la última fila de ResultSet. Se devuelve `false` (falso), independientemente de si el cursor está abierto o cerrado.

DB2BaseDataSource.NO (2)
Para un ResultSet que se ha definido como TYPE_FORWARD_ONLY, se llama al ResultSet.next() y el cursor anteriormente se ha posicionado después de la última fila de ResultSet, el controlador lanza una java.sql.SQLException con el texto de error "Operación no válida: el conjunto de resultados está cerrado". Éste es el valor por omisión.

allowNullResultSetForExecuteQuery
Especifica si IBM Data Server Driver para JDBC y SQLJ devuelve nulo cuando se utiliza Statement.executeQuery, PreparedStatement.executeQuery o CallableStatement.executeQuery para ejecutar una sentencia CALL para un procedimiento almacenado que no devuelve ningún conjunto de resultados.

Los valores posibles son:

DB2BaseDataSource.NOT_SET (0)
El comportamiento es el mismo que para `DB2BaseDataSource.NO`.

DB2BaseDataSource.YES (1)
IBM Data Server Driver para JDBC y SQLJ devuelve nulo cuando se utiliza Statement.executeQuery, PreparedStatement.executeQuery o CallableStatement.executeQuery para ejecutar una sentencia CALL para un procedimiento almacenado que no devuelve ningún conjunto de resultados. Este comportamiento no cumple el estándar de JDBC.

DB2BaseDataSource.NO (2)
IBM Data Server Driver para JDBC y SQLJ emite una SQLException cuando se utiliza Statement.executeQuery, PreparedStatement.executeQuery o CallableStatement.executeQuery para ejecutar una sentencia CALL para un procedimiento almacenado que no devuelve ningún conjunto de resultados. Este comportamiento cumple el estándar de JDBC.

atomicMultiRowInsert
Especifica si las operaciones de proceso por lotes que utilizan métodos PreparedStatement para modificar una tabla son atómicas o no atómicas. El tipo de datos de esta propiedad es int.

En el caso de las conexiones con DB2 para z/OS, esta propiedad se aplica únicamente a las operaciones INSERT de proceso por lotes.

En el caso de las conexiones con DB2 Database para Linux, UNIX y Windows o IBM Informix, esta propiedad se aplica únicamente a las operaciones INSERT, MERGE, UPDATE o DELETE.

Los valores posibles son:

DB2BaseDataSource.YES (1)
Las operaciones de proceso por lotes son atómicas. La inserción de todas las filas en el proceso por lotes se considera una única operación. Si la inserción de una única fila no se realiza con éxito, falla toda la operación con una BatchUpdateException. El uso de una sentencia de proceso por lotes que devuelve claves de generación automática falla con una BatchUpdateException.

Si atomicMultiRowInsert está establecido en `DB2BaseDataSource.YES` (1):

- No se permite la ejecución de sentencias en un lote heterogéneo.
- Si la fuente de datos de destino es DB2 para z/OS no se permiten las operaciones siguientes:
  - La inserción de más de 32.767 filas en un proceso por lotes genera una BatchUpdateException.
  - La llamada a más de uno de los métodos siguientes para el mismo parámetro en filas distintas genera una BatchUpdateException:
    - PreparedStatement.setAsciiStream
    - PreparedStatement.setCharacterStream
    - PreparedStatement.setUnicodeStream

DB2BaseDataSource.NO (2)
Las inserciones de proceso por lotes son no atómicas. La inserción de cada fila se considera una ejecución independiente. La información sobre el éxito de cada operación de inserción procede de la matriz int[] que devuelve Statement.executeBatch.

DB2BaseDataSource.NOT_SET (0)
Las inserciones de proceso por lotes son no atómicas. La inserción de cada fila se considera una ejecución independiente. La información sobre el éxito de cada operación de inserción procede de la matriz int[] que devuelve Statement.executeBatch. Éste es el valor por omisión.

`blockingReadConnectionTimeout`
Cantidad de tiempo en segundos que transcurre hasta que se excede el tiempo de espera de lectura del socket de la conexión. Esta propiedad es aplicable solamente IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4, y afecta a todas las peticiones que se envían a la fuente de datos después de establecer satisfactoriamente una conexión. El valor por omisión es 0, que significa que no hay tiempo de espera.

`clientDebugInfo`
Especifica un valor para el atributo de conexión CLIENT DEBUGINFO. Sirve para notificar al servidor de datos que los procedimientos almacenados y funciones definidas por el usuario que están haciendo uso de la conexión se están ejecutando en la modalidad de depuración. El atributo CLIENT DEBUGINFO es utilizado por el depurador unificado de DB2. El tipo de datos de esta propiedad es String. La longitud máxima es de 254 bytes.

Esta propiedad sólo se aplica a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4.

clientRerouteAlternateServerName
Especifica uno o más nombres de servidores para el redireccionamiento del cliente. El tipo de datos de esta propiedad es String.

Cuando se especifica enableClientAffinitiesList=DB2BaseDataSource.YES (1), clientRerouteAlternateServerName debe contener el nombre del servidor primario y los nombres de los servidores alternativos. El servidor identificado por serverName y portNumber es el servidor primario. Ese nombre de servidor debe aparecer al comienzo de la lista especificada por clientRerouteAlternateServerName.

Si especifica más de un nombre de servidor, delimite los nombres con comas (,) o espacios en blanco. El número de valores especificados para clientRerouteAlternateServerName debe coincidir con el número de valores especificados para clientRerouteAlternatePortNumber.

clientRerouteAlternateServerName es aplicable a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 con DB2 Database para Linux, UNIX y Windows y IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4.

clientRerouteAlternatePortNumber
Especifica uno o más números de puerto para el redireccionamiento del cliente. El tipo de datos de esta propiedad es String.

Cuando se especifica enableClientAffinitiesList=DB2BaseDataSource.YES (1), clientRerouteAlternatePortNumber debe contener el número de puerto del servidor primario y los números de puerto de los servidores alternativos. El servidor identificado por serverName y portNumber es el servidor primario. Ese número de puerto debe aparecer al comienzo de la lista especificada por clientRerouteAlternatePortNumber.

Si especifica más de un número de puerto, delimite los números de puerto con comas (,) o espacios en blanco. El número de valores especificados para clientRerouteAlternatePortNumber debe coincidir con el número de valores especificados para clientRerouteAlternateServerName.

clientRerouteAlternatePortNumber es aplicable a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 con DB2 Database para Linux, UNIX y Windows y IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4.

clientRerouteServerListJNDIName
Identifica una referencia JNDI para una instancia de DB2ClientRerouteServerList en un depósito JNDI de información del servidor de redireccionamiento. La propiedad clientRerouteServerListJNDIName es aplicable solamente a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4, y a las conexiones que se establecen mediante la interfaz DataSource.

Si el valor de clientRerouteServerListJNDIName no es nulo, clientRerouteServerListJNDIName proporcionará las funciones siguientes:
- Permite conservar la información sobre servidores de redireccionamiento al pasar de una JVM a otra
- Proporciona una ubicación de servidor alternativa si falla la primera conexión con la fuente de datos

clientRerouteServerListJNDIContext

Especifica el contexto de JNDI que se utiliza para la vinculación y la consulta de la instancia de DB2ClientRerouteServerList. La propiedad clientRerouteServerListJNDIContext es aplicable solamente a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4, y a las conexiones que se establecen mediante la interfaz DataSource.

Si no se define clientRerouteServerListJNDIContext, IBM Data Server Driver para JDBC y SQLJ creará un contexto inicial utilizando propiedades del sistema o el archivo jndi.properties.

clientRerouteServerListJNDIContext **sólo** se puede definir mediante el método siguiente:

```
public void setClientRerouteServerListJNDIContext(javax.naming.Context registro)
```

commandTimeout

Especifica el tiempo máximo en segundos que una aplicación que se ejecuta bajo IBM Data Server Driver para JDBC y SQLJ espera a que se complete cualquier tipo de petición a un grupo de servidores de datos antes de que el controlador emita una SQLException. El tiempo de espera incluye el tiempo para obtener un transporte, realizar la migración tras error si es necesario, enviar la petición y esperar una respuesta. El tipo de datos de este parámetro es int. El valor por omisión es 0.

commandTimeout se aplica únicamente a las conexiones con un grupo de compartimiento de datos de DB2 para z/OS, una instancia DB2 pureScale o un clúster de alta disponibilidad de IBM Informix.

Si se invoca el método java.sql.Statement.setQueryTimeout, el valor de tiempo de espera de consulta que se establece a través de Statement.setQueryTimeout altera temporalmente el valor commandTimeout.

El código de error de SQL que se devuelve con SQLException depende del servidor de datos y del valor de la propiedad queryTimeoutInterruptProcessingMode:

- Para las conexiones con servidores de datos DB2 para z/OS, se devuelve -30108.
- Para las conexiones con otros servidores de datos:
  - Si el valor de queryTimeoutInterruptProcessingMode es INTERRUPT_PROCESSING_MODE_STATEMENT_CANCEL (1), se devuelve -952.
  - Si el valor de queryTimeoutInterruptProcessingMode es INTERRUPT_PROCESSING_MODE_CLOSE_SOCKET (2), se devuelve -30108.

Si la propiedad de configuración db2.jcc.enableInetAddressGetHostName se establece en true, podrían producirse las situaciones siguientes:

- Los tiempos de espera reales exceden el valor commandTimeout. Esta situación puede producirse cuando el controlador necesita realizar varias operaciones de búsqueda de DNS para resolver direcciones IP para nombres de sistema principal. La cantidad en que el tiempo de espera excede el valor commandTimeout depende del número de operaciones de búsqueda de DNS y la cantidad de tiempo que cada operación de búsqueda de DNS emplea.
- El tiempo adicional que se requiere para las operaciones de búsqueda de DNS podría generar más condiciones de tiempo de espera excedido que si db2.jcc.enableInetAddressGetHostName se establece en `false`.

connectionCloseWithInFlightTransaction

Especifica si IBM Data Server Driver para JDBC y SQLJ emite una SQLException o retrotrae una transacción sin emitir una SQLException cuando se cierra una conexión en mitad de la transacción. Los valores posibles son:

DB2BaseDataSource.NOT_SET (0)
El comportamiento es el mismo que para DB2BaseDataSource.CONNECTION_CLOSE_WITH_EXCEPTION.

DB2BaseDataSource.CONNECTION_CLOSE_WITH_EXCEPTION (1)
Cuando se cierra una conexión en medio de una transacción, se emite una SQLException con el error -4471.

DB2BaseDataSource.CONNECTION_CLOSE_WITH_ROLLBACK (2)
Cuando se cierra una conexión en medio de una transacción, la transacción se retrotrae y no se emite ninguna SQLException.

connectionTimeout

Especifica el tiempo máximo en segundos que IBM Data Server Driver para JDBC y SQLJ espera una respuesta de un grupo de servidores de datos cuando el controlador intenta establecer una conexión. Si el controlador no recibe una respuesta después de la cantidad de tiempo que se ha especificado mediante connectionTimeout, emite una SQLException con el código de error de SQL -4499. El tipo de datos de este parámetro es int. El valor por omisión es 0.

connectionTimeout se aplica únicamente a las conexiones con un grupo de compartimiento de datos de DB2 para z/OS, una instancia DB2 pureScale o un clúster de alta disponibilidad de IBM Informix.

Si connectionTimeout se establece en un valor positivo, ese valor prevalece sobre cualquier otro valor de tiempo de espera que se haya establecido en una conexión, como loginTimeout. Se intenta establecer una conexión con el miembro del grupo de servidores de datos que tenga la mayor capacidad de carga. Si ninguno de los miembros está activo, se intenta establecer una conexión con la dirección IP de grupo especificada en DataSource. Si la conexión no se puede establecer con ninguno de los servidores de datos dentro del plazo de tiempo especificado mediante connectionTimeout, se emite una SQLException.

Si connectionTimeout se establece en 0, y el redireccionamiento automático de cliente no está habilitado, no existirá ningún límite de tiempo.

Si connectionTimeout se establece en 0, y el redireccionamiento automático de cliente está habilitado para un grupo de compartimiento de datos de DB2 para z/OS, una instancia DB2 pureScale o un clúster de alta disponibilidad de IBM Informix, las propiedades del redireccionamiento automático de cliente como maxRetriesForClientReroute y retryIntervalForClientReroute controlarán la cantidad de tiempo necesaria para establecer la conexión.

Si la propiedad de configuración db2.jcc.enableInetAddressGetHostName se establece en true, podrían producirse las situaciones siguientes:

- Los tiempos de espera reales exceden el valor connectionTimeout. Esta situación puede producirse cuando el controlador necesita realizar varias operaciones de búsqueda de DNS para resolver direcciones IP para nombres de sistema principal. La cantidad en que el tiempo de espera excede el valor connectionTimeout depende del número de operaciones de búsqueda de DNS y la cantidad de tiempo que cada operación de búsqueda de DNS emplea.

- El tiempo adicional que se requiere para las operaciones de búsqueda de DNS podría generar más condiciones de tiempo de espera excedido que si db2.jcc.enableInetAddressGetHostName se establece en `false`.

databaseName

Especifica el nombre de la fuente de datos. Este nombre constituye la parte del URL de conexión correspondiente a la *basedatos*. El nombre depende de si se utiliza IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 o IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2.

Para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4:

- Si la conexión es con un servidor DB2 para z/OS, el valor de databaseName es el nombre de ubicación de DB2 que se define durante la instalación. Todos los caracteres de este valor deben ser caracteres en mayúsculas. Puede determinar el nombre de ubicación ejecutando la sentencia de SQL siguiente en el servidor:

  ```
  SELECT CURRENT SERVER FROM SYSIBM.SYSDUMMY1;
  ```

- Si la conexión es con un servidor DB2 Database para Linux, UNIX y Windows, el valor de databaseName será el nombre de la base de datos que se define durante la instalación.
- Si la conexión es con un servidor IBM Informix, *basedatos* es el nombre de la base de datos. El nombre no es sensible a las mayúsculas y las minúsculas. El servidor convierte el nombre a minúsculas.
- Si la conexión es con un servidor IBM Cloudscape, el valor de databaseName será el nombre totalmente calificado del archivo que contiene la base de datos. Este nombre se debe incluir entre comillas dobles ("). Por ejemplo:

  ```
  "c:/basedatos/testdb"
  ```

Si esta propiedad no está definida, las conexiones se establecen con el sitio local.

Para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2:

- El valor de databaseName es el nombre de base de datos que se define durante la instalación, si el valor de la propiedad de conexión serverName es nulo. Si el valor de la propiedad serverName no es nulo, el valor de databaseName es un alias de base de datos.

decimalSeparator

Especifica el separador decimal para valores de datos de entrada y salida, valores decimales, valores de coma flotante o valores de coma flotante decimal. El tipo de datos de esta propiedad es int.

Si el valor de la propiedad sendDataAsIs es `true`, decimalSeparator solo afecta a los valores de salida.

Los valores posibles son:

DB2BaseDataSource.DECIMAL_SEPARATOR_NOT_SET (0)
Se utiliza un punto como separador decimal. Éste es el valor por omisión.

DB2BaseDataSource.DECIMAL_SEPARATOR_PERIOD (1)
Se utiliza un punto como separador decimal.

DB2BaseDataSource.DECIMAL_SEPARATOR_COMMA (2)
Se utiliza una coma como separador decimal.

Cuando se establece DECIMAL_SEPARATOR_COMMA, el resultado de ResultSet.getString en un valor decimal, de coma flotante o de coma

flotante decimal tiene una coma como separador. Sin embargo, si el método toString se ejecuta en un valor que se recupera con un método ResultSet.getXXX que devuelve un valor decimal, de coma flotante o de coma flotante decimal, el resultado tiene una coma decimal como separador decimal.

decimalStringFormat
Especifica el formato de serie de los datos que se recuperan de una columna DECIMAL o DECFLOAT cuando la versión de SDK para Java es la Versión 1.5 o posterior. El tipo de datos de esta propiedad es int. Los valores posibles son:

DB2BaseDataSource.DECIMAL_STRING_FORMAT_NOT_SET (0)
IBM Data Server Driver para JDBC y SQLJ devuelve los valores decimales en el formato en el que los devuelve el método java.math.BigDecimal.toString. Éste es el valor por omisión.

Por ejemplo, el valor 0,0000000004 se devuelve como 4E-10.

DB2BaseDataSource.DECIMAL_STRING_FORMAT_TO_STRING (1)
IBM Data Server Driver para JDBC y SQLJ devuelve los valores decimales en el formato en el que los devuelve el método java.math.BigDecimal.toString.

Por ejemplo, el valor 0,0000000004 se devuelve como 4E-10.

DB2BaseDataSource.DECIMAL_STRING_FORMAT_TO_PLAIN_STRING (2)
IBM Data Server Driver para JDBC y SQLJ devuelve valores decimales en el formato en que los devuelve el método java.math.BigDecimal.toPlainString.

Por ejemplo, el valor 0,0000000004 se devuelve como 0,0000000004.

Esta propiedad no tiene efecto sobre las versiones anteriores de SDK para Java. Para esas versiones, IBM Data Server Driver para JDBC y SQLJ devuelve valores decimales en el formato en el que los devuelve el método java.math.BigDecimal.toString.

defaultIsolationLevel
Especifica el nivel de aislamiento por omisión de las transacciones para las nuevas conexiones. El tipo de datos de esta propiedad es int. Cuando se define defaultIsolationLevel para un DataSource, todas las conexiones creadas desde ese DataSource tienen el nivel de aislamiento por omisión especificado por defaultIsolationLevel.

Para las fuentes de datos DB2, el valor por omisión es `java.sql.Connection.TRANSACTION_READ_COMMITTED`.

Para las bases de datos IBM Informix, el valor por omisión depende del tipo de fuente de datos. La tabla siguiente muestra los valores por omisión.

Tabla 54. Niveles de aislamiento por omisión para bases de datos IBM Informix

Tipo de fuente de datos	Nivel de aislamiento por omisión
Base de datos compatible con ANSI con anotación cronológica	java.sql.Connection.TRANSACTION_SERIALIZABLE
Base de datos sin anotación cronológica	java.sql.Connection.TRANSACTION_READ_UNCOMMITTED
Base de datos no compatible con ANSI con anotación cronológica	java.sql.Connection.TRANSACTION_READ_COMMITTED

deferPrepares
Especifica si la invocación del método Connection.prepareStatement produce la

preparación inmediata de una sentencia de SQL en la fuente de datos o si la preparación de la sentencia se aplaza hasta que se ejecuta el método PreparedStatement.execute. El tipo de datos de esta propiedad es booleano.

deferPrepares se puede utilizar para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 con DB2 Database para Linux, UNIX y Windows, y IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4.

Los valores posibles son:

true La preparación de la sentencia en la fuente de datos no se produce hasta que se ejecuta el método PreparedStatement.execute. Éste es el valor por omisión.

false La preparación de la sentencia en la fuente de datos se produce cuando se ejecuta el método Connection.prepareStatement.

El aplazar las operaciones de preparación puede reducir los retardos de la red. Pero si aplaza operaciones de preparación, debe asegurarse de que los tipos de datos de entrada coinciden con los tipos de columnas de tablas.

descripción
Descripción de la fuente de datos. El tipo de datos de esta propiedad es String.

downgradeHoldCursorsUnderXa
Especifica si los cursores que se han definido como WITH HOLD pueden abrirse en conexiones XA.

downgradeHoldCursorsUnderXa se aplica a:

- IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 con servidores DB2 para z/OS.
- IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 o IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 con servidores DB2 Database para Linux, UNIX y Windows.

El valor por omisión es `false`, que significa que no se puede abrir un cursor definido con WITH HOLD en una conexión XA. Se emite una excepción cuando se realiza un intento de abrir ese cursor.

Si downgradeHoldCursorsUnderXa se establece en `true`, un cursor definido con WITH HOLD puede abrirse en una conexión XA. Sin embargo, el cursor tiene las restricciones siguientes:

- Cuando el cursor se abre en una conexión XA, el cursor no tiene el comportamiento con WITH HOLD. El cursor se cierra en XA End.
- Un cursor abierto antes de XA Start en una transacción local se cierra al realizar la operación XA Start.

driverType
Para la interfaz DataSource, determina qué controlador utilizar para las conexiones. El tipo de datos de esta propiedad es int. Los valores válidos son 2 o 4. El valor por omisión es 2.

enableClientAffinitiesList
Especifica si IBM Data Server Driver para JDBC y SQLJ permite las afinidades de cliente para el soporte de migración tras error en cascada. El tipo de datos de esta propiedad es int. Los valores posibles son:

DB2BaseDataSource.YES (1)
IBM Data Server Driver para JDBC y SQLJ permite las afinidades de cliente para el soporte de migración tras error en cascada. Esto significa que el reintento de conexión solamente se realiza para los servidores

especificados en las propiedades clientRerouteAlternateServerName y clientRerouteAlternatePortNumber. El controlador no intenta volver a establecer la conexión con ningún otro servidor.

Por ejemplo, suponga que clientRerouteAlternateServerName contiene la siguiente serie:

```
host1,host2,host3
```

Supongamos también que clientRerouteAlternatePortNumber contiene la serie siguiente:

```
port1,port2,port3
```

Cuando las afinidades de cliente están habilitadas, el orden de reintento es:

1. host1:port1
2. host2:port2
3. host3:port3

DB2BaseDataSource.NO (2)
IBM Data Server Driver para JDBC y SQLJ no permite las afinidades de clienta para el soporte de migración tras error en cascada.

DB2BaseDataSource.NOT_SET (0)
IBM Data Server Driver para JDBC y SQLJ no permite las afinidades de clienta para el soporte de migración tras error en cascada. Éste es el valor por omisión.

El efecto de las propiedades maxRetriesForClientReroute y retryIntervalForClientReroute difiere dependiendo de si está o no habilitado enableClientAffinitiesList.

Esta propiedad sólo se aplica a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4.

enableNamedParameterMarkers
Especifica si el soporte para los marcadores de parámetro con nombre está habilitado en IBM Data Server Driver para JDBC y SQLJ. El tipo de datos de esta propiedad es int. Los valores posibles son:

DB2BaseDataSource.YES (1)
El soporte de marcador de parámetro con nombre está habilitado en IBM Data Server Driver para JDBC y SQLJ.

DB2BaseDataSource.NO (2)
El soporte para marcadores de parámetro con nombre no está habilitado en IBM Data Server Driver para JDBC y SQLJ.

El controlador envía una sentencia de SQL con marcadores de parámetro con nombre a la fuente de datos de destino sin modificación. El resultado satisfactorio o anómalo de la sentencia depende de una serie de factores, incluidos los siguientes:

- Si la fuente de datos de destino da soporte a los marcadores de parámetro con nombre
- Si el valor de la propiedad deferPrepares es true o false
- Si el valor de la propiedad sendDataAsIs es true o false

Recomendación: Para evitar un comportamiento inesperado en una aplicación que utiliza marcadores de parámetro con nombre, establezca enableNamedParameterMarkers en YES.

DB2BaseDataSource.NOT_SET (0)
El comportamiento es el mismo que el de DB2BaseDataSource.NO (2). Éste es el valor por omisión.

enableSeamlessFailover
Especifica si IBM Data Server Driver para JDBC y SQLJ utiliza la migración tras error sin fisuras para el redireccionamiento del cliente. El tipo de datos de esta propiedad es int.

Para las conexiones con DB2 para z/OS, si enableSysplexWLB está definido en true, enableSeamlessFailover no tiene ningún efecto. IBM Data Server Driver para JDBC y SQLJ utiliza migración tras error sin fisuras independientemente del valor de enableSeamlessFailover.

Los valores posibles de enableSeamlessFailover son:

DB2BaseDataSource.YES (1)
IBM Data Server Driver para JDBC y SQLJ utiliza la migración tras error sin fisuras. Esto significa que el controlador no lanzará una SQLException con el código de error de SQL -4498 tras restablecer satisfactoriamente una conexión fallida si se cumplen las siguientes condiciones:

- La conexión no estaba siendo utilizada para una transacción en el momento en el que se ha producido la anomalía.
- No existen recursos globales pendientes, tales como tablas temporales globales, cursores retenidos abiertos o estados de conexión que eviten una migración tras error sin fisuras con otro servidor.

Si se produce una migración tras error sin fisuras, después de establecer la conexión con una nueva fuente de datos, el controlador vuelve a lanzar la sentencia de SQL que estaba siendo procesada en el momento que ha fallado la conexión original.

Recomendación: establezca la propiedad queryCloseImplicit en DB2BaseDataSource.QUERY_CLOSE_IMPLICIT_NO (2) cuando fije enableSeamlessFailover en DB2BaseDataSource.YES, si la aplicación utiliza cursores retenidos.

DB2BaseDataSource.NO (2)
IBM Data Server Driver para JDBC y SQLJ no utiliza la migración tras error sin fisuras.

Cuando este valor está en vigor, si un servidor pasa a estar inactivo, el controlador intenta transferir la conexión a un servidor alternativo. Si se realiza correctamente una migración tras error o una recuperación, el controlador emite una SQLException con el código de error de SQL -4498, el cual indica que ha fallado una conexión, pero se ha restablecido satisfactoriamente. Una SQLException con el código de error de SQL -4498 informa a la aplicación de que debe reintentar la transacción durante la cual se produjo el error de conexión. Si el controlador no puede restablecer una conexión, emite una SQLException con el código de error de SQL -4499.

DB2BaseDataSource.NOT_SET (0)
IBM Data Server Driver para JDBC y SQLJ no utiliza la migración tras error sin fisuras. Éste es el valor por omisión.

enableSysplexWLB
Indica si la función de equilibrado de carga de trabajo Sysplex de IBM Data

Server Driver para JDBC y SQLJ está habilitada. El tipo de datos de enableSysplexWLB es booleano. El valor por omisión es `false`.

enablSysplexWLB se aplica únicamente a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4.

fetchSize
Especifica el tamaño de captación por omisión para objetos ResultSet que se han generado a partir de objetos Statement. El tipo de datos de esta propiedad es int.

El valor por omisión de fetchSize puede verse alterado temporalmente por el método Statement.setFetchSize. La propiedad fetchSize no afecta a los objetos Statement que ya existían cuando se estableció fetchSize.

Los valores posibles de fetchSize son:

0 o *entero positivo*
El valor por omisión de *fetchSize* para los objetos Statement recién creados. Si el valor de propiedad de fetchSize no es válido, IBM Data Server Driver para JDBC y SQLJ establecer el valor por omisión de *fetchSize* en 0.

DB2BaseDataSource.FETCHSIZE_NOT_SET (-1)
Indica que el valor por omisión de *fetchSize* para los objetos Statement es 0. Se trata de la propiedad por omisión.

La propiedad fetchSize difiere de la propiedad queryDataSize: fetchSize afecta al número de filas que se devuelven y queryDataSize afecta al número de bytes que se devuelven.

fullyMaterializeLobData
Indica si el controlador recupera localizadores de LOB para operaciones FETCH. El tipo de datos de esta propiedad es booleano.

El efecto de fullyMaterializeLobData depende de si la fuente de datos es compatible con la modalidad continua progresiva, conocida también como formato de datos dinámico:

- Si la fuente de datos no es compatible con la modalidad continua progresiva:

  Si el valor de fullyMaterializeLobData es `true`, los datos LOB se materializarán completamente dentro del controlador JDBC cuando se capte una fila. Si el valor es `false`, los datos LOB se canalizarán. El controlador utiliza localizadores internamente para recuperar datos LOB en forma de bloques a medida que sea necesario. Es muy recomendable que establezca este valor en `false` cuando recupere datos LOB que contengan grandes volúmenes de datos. El valor por omisión es `true`.
- Si la fuente de datos es compatible con la modalidad continua progresiva:

  El controlador JDBC no tiene en cuenta el valor de fullyMaterializeLobData si la propiedad progressiveStreaming está establecida en `DB2BaseDataSource.YES` o `DB2BaseDataSource.NOT_SET`.

Esta propiedad no afecta a parámetros de procedimiento almacenado ni a los LOB que se recuperan mediante cursores desplazables. Los parámetros de procedimiento almacenado de LOB se materializan siempre. Los LOB que se recuperan utilizando cursores desplazables utilizan localizadores de LOB si la modalidad continua progresiva no está en vigor.

implicitRollbackOption
Especifica las acciones que IBM Data Server Driver para JDBC y SQLJ llevará a

cabo cuando una transacción encuentra un punto muerto o un tiempo de espera excedido. Los valores posibles son:

DB2BaseDataSource.IMPLICIT_ROLLBACK_OPTION_NOT_CLOSE_CONNECTION (1)
IBM Data Server Driver para JDBC y SQLJ emite una SQLException con un código de error de SQL que indica que se ha producido un punto muerto o un tiempo de espera excedido. El código de error de SQL es el código de error de SQL generado por el servidor de datos después de un punto muerto o un tiempo de espera excedido. El controlador no cierra la conexión.

DB2BaseDataSource.IMPLICIT_ROLLBACK_OPTION_CLOSE_CONNECTION (2)
IBM Data Server Driver para JDBC y SQLJ emite una DisconnectException con el código de error de SQL -4499 cuando se produce un punto muerto o un tiempo de espera excedido. El controlador cierra la conexión. Si el redireccionamiento automático de cliente o el equilibrado de la carga de trabajo de Sysplex está habilitado, el controlador inhabilita el comportamiento de migración automática tras error.

DB2BaseDataSource.IMPLICIT_ROLLBACK_OPTION_NOT_SET (0)
Éste es el valor por omisión. IBM Data Server Driver para JDBC y SQLJ emite una SQLException con un código de error de SQL que indica que se ha producido un punto muerto o un tiempo de espera excedido. El código de error de SQL es el código de error de SQL generado por el servidor de datos después de un punto muerto o un tiempo de espera excedido. El controlador no cierra la conexión.

interruptProcessingMode
Especifica el comportamiento de IBM Data Server Driver para JDBC y SQLJ cuando una aplicación ejecuta el método Statement.cancel. Los valores posibles son:

DB2BaseDataSource.INTERRUPT_PROCESSING_MODE_DISABLED (0)
El proceso de interrupción está inhabilitado. Cuando una aplicación ejecuta Statement.cancel, IBM Data Server Driver para JDBC y SQLJ no realiza ninguna acción.

DB2BaseDataSource.INTERRUPT_PROCESSING_MODE_STATEMENT_CANCEL (1)
Cuando una aplicación ejecuta Statement.cancel, IBM Data Server Driver para JDBC y SQLJ cancela la sentencia que se está ejecutando actualmente, siempre y cuando el servidor de datos soporte el proceso de interrupciones. Si el servidor de datos no da soporte al proceso de interrupciones, IBM Data Server Driver para JDBC y SQLJ emite una SQLException que indica que la función no está soportada. INTERRUPT_PROCESSING_MODE_STATEMENT_CANCEL es el valor por omisión.

Para los clientes de DB2 Database para Linux, UNIX y Windows, cuando interruptProcessingMode se establece en INTERRUPT_PROCESSING_MODE_STATEMENT_CANCEL, el valor de DB2 Connect para INTERRUPT_ENABLED y el valor de la variable de registro de DB2 para DB2CONNECT_DISCONNECT_ON_INTERRUPT sustituyen a este valor.

DB2BaseDataSource.INTERRUPT_PROCESSING_MODE_CLOSE_SOCKET (2)
Cuando una aplicación ejecuta Statement.cancel, IBM Data Server Driver para JDBC y SQLJ descarta el socket subyacente. La conexión

no se cierra y puede reutilizarse para volver a someter la sentencia. Cuando la conexión vuelve a utilizarse, el controlador obtiene un nuevo socket.

Para las conexiones con servidores de datos DB2 para z/OS, IBM Data Server Driver para JDBC y SQLJ siempre utiliza este valor, con independencia del valor que se haya especificado.

keepAliveTimeout
El tiempo máximo en segundos previo a que cada señal TCP KeepAlive se envíe al servidor de datos. El tipo de datos de esta propiedad es int. El valor por omisión es 15 segundos.

IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 utiliza el protocolo TCP/IP para comunicarse con los servidores de datos. Para evitar problemas potenciales de migración tras error causados por los tiempos de espera excedidos dentro de la capa TCP/IP, es necesario ajustar los parámetros TCP/IP KeepAlive en el cliente. La disminución de los valores KeepAlive en el cliente mejora detección oportuna de anomalías del servidor.

keepAliveTimeout solo recibe soporte para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4.

loginTimeout
Cantidad máxima de tiempo, en segundos, que se debe esperar para establecer conexión con una fuente de datos. Una vez transcurrido el número de segundos especificado por loginTimeout, el controlador cierra la conexión con la fuente de datos. El tipo de datos de esta propiedad es int. El valor por omisión es 0, que es el valor por omisión del sistema para el tiempo de espera. Esta propiedad no está soportada para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS.

Si el entorno de servidor de datos es un entorno de equilibrado de carga de trabajo de Sysplex de DB2 para z/OS o un entorno DB2 pureScale, el tiempo de espera para una conexión viene determinado por una combinación de loginTimeout, maxRetriesForClientReroute y retryIntervalForClientReroute. loginTimeout determina el tiempo para un solo intento de establecer una conexión con un servidor de datos. Pueden haber varios intentos de establecer una conexión, según el valor de maxRetriesForClientReroute. Asimismo, pueden haber vacíos entre intentos de establecer una conexión, según el valor de retryIntervalForClientReroute.

logWriter
Es la corriente de salida donde se escriben todos los mensajes de anotaciones cronológicas y de rastreo correspondientes al objeto DataSource. El tipo de datos de esta propiedad es java.io.PrinterWriter. El valor por omisión es nulo, que significa que no se generan datos de anotaciones cronológicas ni de rastreo para DataSource.

maxRetriesForClientReroute
Durante el redireccionamiento automático del cliente, limita el número de reintentos si falla la conexión primaria con la fuente de datos.

El tipo de datos de esta propiedad es int.

IBM Data Server Driver para JDBC y SQLJ utiliza la propiedad maxRetriesForClientReroute solamente si también establecida la propiedad retryIntervalForClientReroute.

Si el valor de enableClientAffinitiesList es DB2BaseDataSource.NO (2), el intento de conectar con el servidor primario y los servidores alternativos se

contabiliza como un reintento. Si el valor de enableClientAffinitiesList es DB2BaseDataSource.YES (1), se intenta conectar con cada servidor especificado por los valores de clientRerouteAlternateServerName y clientRerouteAlternatePortNumber, durante el número de veces especificado por maxRetriesForClientReroute.

El valor por omisión de maxRetriesForClientReroute viene determinado de la siguiente forma:

- Si enableClientAffinitiesList es DB2BaseDataSource.YES (1), el valor por omisión es 0.
- Para las conexiones con los servidores de datos DB2 Database para Linux, UNIX y Windows o IBM Informix, si no se han establecido maxRetriesForClientReroute ni retryIntervalForClientReroute, la conexión vuelve a intentarse durante 10 minutos, con un tiempo de espera entre reintentos que se incrementa a medida que incrementa el periodo de tiempo desde el primer reintento.
- Para las conexiones con otros servidores de datos DB2 para z/OS:
  - Para la Versión 3.63, 4.13 o posterior de IBM Data Server Driver para JDBC y SQLJ:
    - Si la propiedad enableSysplexWLB se ha establecido en `true` y no se han establecido maxRetriesForClientReroute ni retryIntervalForClientReroute, el valor por omisión es 5.
    - Si la propiedad enableSysplexWLB se ha establecido en `false` y no se han establecido maxRetriesForClientReroute ni retryIntervalForClientReroute, la conexión vuelve a intentarse durante 10 minutos, con un tiempo de espera entre reintentos que se incrementa a medida que incrementa el periodo de tiempo desde el primer reintento.
  - Para las versiones de IBM Data Server Driver para JDBC y SQLJ anteriores a la 3.63 o la 4.13, si no se ha establecido maxRetriesForClientReroute ni retryIntervalForClientReroute, la conexión se reintenta durante 10 minutos, con un tiempo de espera entre reintentos que aumenta a medida que se incrementa el tiempo transcurrido desde el primer reintento.

Si el valor de maxRetriesForClientReroute es 0, no se produce el proceso de redireccionamiento del cliente.

maxStatements

Controla una antememoria interna de sentencias que está asociada a un Connection. El tipo de datos de esta propiedad es int. Los valores posibles son:

entero positivo
Habilita la antememoria interna de sentencias para Connection y especifica el número de sentencias que IBM Data Server Driver para JDBC y SQLJ mantiene abiertas en la antememoria.

0 o entero negativo
Inhabilita el almacenamiento en antememoria interna de sentencias para el objeto Connection. 0 es el valor por omisión.

com.ibm.db2.jcc.DB2SimpleDataSource.maxStatements controla la antememoria interna de sentencias que se asocia con una Connection solamente cuando se crea el objeto Connection.
com.ibm.db2.jcc.DB2SimpleDataSource.maxStatements no tiene ningún efecto sobre el almacenamiento en antememoria para un objeto Connection ya existente.

com.ibm.db2.jcc.DB2SimpleDataSource.maxStatements solo se aplica a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4.

memberConnectTimeout
Especifica la cantidad de tiempo en segundos antes de que falle un intento de abrir un socket a un grupo de compartimiento de datos de DB2 para z/OS, una instancia DB2 pureScale o un clúster de alta disponibilidad de IBM Informix. Este valor de tiempo de espera sólo se aplica a los intentos de conexión del socket con diferentes miembros durante una migración tras error. El tipo de datos de esta propiedad es int.

Para las conexiones con servidores de datos DB2 para z/OS, el valor por omisión es un segundo. Para las conexiones con otros servidores de datos, el valor por omisión es 0.

Si el valor memberConnectTimeout es igual o inferior a 0, el controlador utiliza el valor loginTimeout para determinar cuánto tiempo debe esperar antes de que falle la petición de conexión.

El valor memberConnectTimeout se utiliza para cada operación de apertura de socket de cada uno de los miembros de la lista.

En el caso de una conexión con un grupo de compartimiento de datos de DB2 para z/OS, después de que hayan fallado todos los intentos de abrir un socket a todos los miembros, el controlador vuelve a intentar abrir el socket utilizando una dirección IP de grupo. Para ese reintento, el controlador utiliza el valor loginTimeout para determinar cuánto tiempo debe esperar antes de que falle la petición de conexión.

contraseña
Es la contraseña que se debe utilizar para establecer conexiones. El tipo de datos de esta propiedad es String. Cuando utiliza la interfaz DataSource para establecer una conexión, puede alterar temporalmente el valor de esta propiedad invocando esta modalidad del método DataSource.getConnection:

```
getConnection(usuario, contraseña);
```

portNumber
El número de puerto en que el servidor DRDA escucha peticiones. El tipo de datos de esta propiedad es int.

progressiveStreaming
Especifica si el controlador JDBC utiliza la modalidad continua progresiva cuando esta función es compatible con la fuente de datos.

DB2 para z/OS Versión 9.1 y versiones posteriores es compatible con la modalidad continua progresiva para objetos LOB y XML. DB2 Database para Linux, UNIX y Windows Versión 9.5 y posteriores, e IBM Informix Versión 11.50 y posteriores soportan la modalidad continua progresiva para los LOB.

Cuando se utiliza la modalidad continua progresiva, también conocida como formato de datos dinámico, la fuente de datos determina dinámicamente la forma más eficiente de devolver datos LOB o XML, de acuerdo con el tamaño de los objetos LOB o XML. El valor del parámetro streamBufferSize determina si los datos se materializan cuando se devuelven.

El tipo de datos de progressiveStreaming es int. Los valores válidos son `DB2BaseDataSource.YES` (1) y `DB2BaseDataSource.NO` (2). Si la propiedad progressiveStreaming no está especificada, el valor de progressiveStreaming es `DB2BaseDataSource.NOT_SET` (0).

Si la conexión es con una fuente de datos que es compatible con la modalidad continua progresiva, y el valor de progressiveStreaming es

DB2BaseDataSource.YES o DB2BaseDataSource.NOT_SET, el controlador JDBC utiliza la modalidad continua progresiva para devolver datos de LOB y XML.

Si el valor de progressiveStreaming es DB2BaseDataSource.NO, o la fuente de datos no es compatible con la modalidad continua progresiva, la forma en que el controlador JDBC devuelve datos de LOB o XML depende del valor de la propiedad fullyMaterializeLobData.

queryCloseImplicit

Especifica si los cursores se cierran inmediatamente después de que se hayan captado todas las filas. queryCloseImplicit se aplica únicamente a las conexiones a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 para DB2 para z/OS Versión 8 o posteriores, y IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 o IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2DB2 Database para Linux, UNIX y Windows Versión 9.7 o posterior. Los valores posibles son:

DB2BaseDataSource.QUERY_CLOSE_IMPLICIT_YES (1)

Cerrar los cursores inmediatamente después de que se hayan captado todas las filas.

Un valor de DB2BaseDataSource.QUERY_CLOSE_IMPLICIT_YES puede tener un rendimiento mejor, ya que esta configuración reduce el tráfico de red.

DB2BaseDataSource.QUERY_CLOSE_IMPLICIT_NO (2)

No cerrar los cursores inmediatamente después de que se hayan captado todas las filas.

DB2BaseDataSource.QUERY_CLOSE_IMPLICIT_COMMIT (3)

Realizar las acciones siguientes:

- Cerrar el cursor implícitamente después de que se hayan captado todas las filas.
- Si la aplicación se encuentra en modalidad de confirmación automática, enviar implícitamente una petición de confirmación a la fuente de datos de la unidad de trabajo actual.

Importante: El establecimiento de este valor podría influir en otros recursos, del mismo modo que una operación de confirmación explícita podría influir en otros recursos. Por ejemplo, se cierran otros cursores no retenidos, los localizadores de LOB quedan fuera del ámbito, se restauran las referencias progresivas y los cursores desplazables pierden su posición.

Restricción: Se aplican las restricciones siguientes al comportamiento QUERY_CLOSE_IMPLICIT_COMMIT:

- Este comportamiento se aplica únicamente a las sentencias SELECT que emite la aplicación. No se aplica a las sentencias SELECT que genera IBM Data Server Driver para JDBC y SQLJ.
- Si se ha determinado QUERY_CLOSE_IMPLICIT_COMMIT y la aplicación no se encuentra en modalidad de confirmación automática, el controlador utiliza el comportamiento por omisión (comportamiento QUERY_CLOSE_IMPLICIT_NOT_SET). Si QUERY_CLOSE_IMPLICIT_COMMIT es el comportamiento por omisión, el controlador emplea el comportamiento QUERY_CLOSE_IMPLICIT_YES.
- Si se ha establecido QUERY_CLOSE_IMPLICIT_COMMIT y la fuente de datos no da soporte al comportamiento

QUERY_CLOSE_IMPLICIT_COMMIT, el controlador utiliza el comportamiento QUERY_CLOSE_IMPLICIT_YES.

- Este comportamiento no recibe soporte para las sentencias de proceso por lotes.
- Se da soporte a este comportamiento en una conexión XA únicamente cuando la conexión se encuentra en una transacción local.

DB2BaseDataSource.QUERY_CLOSE_IMPLICIT_NOT_SET (0)
Éste es el valor por omisión. La tabla siguiente describe el comportamiento para una conexión con cada tipo de fuente de datos.

Fuente de datos	Version	Entorno de compartición de datos	Comportamiento
DB2 para z/OS	Versión 10	Entorno de compartición de datos o sin compartición de datos	QUERY_CLOSE_IMPLICIT_COMMIT
DB2 para z/OS	Versión 9 con APAR PK68746	Sin compartición de datos, o en un grupo de compartimiento de datos, pero no en modalidad de coexistencia con miembros de la Versión 8	QUERY_CLOSE_IMPLICIT_COMMIT
DB2 para z/OS	Versión 9 sin APAR PK68746	Sin compartición de datos, o en un grupo de compartimiento de datos, pero no en modalidad de coexistencia con miembros de la Versión 8	QUERY_CLOSE_IMPLICIT_YES
DB2 para z/OS	Versión 9 con APAR PK68746	En un grupo de compartimiento de datos en modalidad de coexistencia con miembros de la Versión 8	QUERY_CLOSE_IMPLICIT_COMMIT
DB2 para z/OS	Versión 9 sin APAR PK68746	En un grupo de compartimiento de datos en modalidad de coexistencia con miembros de la Versión 8	QUERY_CLOSE_IMPLICIT_YES
DB2 para z/OS	Versión 8 con o sin APAR PK68746		QUERY_CLOSE_IMPLICIT_YES
DB2 Database para Linux, UNIX y Windows	Versión 9.7		QUERY_CLOSE_IMPLICIT_YES

queryDataSize
Especifica un valor para controlar la cantidad de datos de una consulta, en bytes, que son devueltos por la fuente de datos en cada operación de recuperación de datos. Este valor se puede utilizar para optimizar la aplicación mediante el control del número de veces que es necesario acceder a la fuente de datos para recuperar los datos.

Un valor mayor de queryDataSize puede originar un menor tráfico de red, y como resultado puede mejorar el rendimiento. Por ejemplo, si el tamaño del conjunto de resultados es 50 KB, y el valor de queryDataSize es 32767 (32KB), son necesarios dos accesos al servidor de bases de datos para recuperar el conjunto de resultados. Sin embargo, si queryDataSize se ha establecido en 65535 (64 KB), solo es necesario acceder una única vez a la fuente de datos para recuperar el conjunto de resultados.

La tabla siguiente muestra los valores mínimo, máximo y por omisión de queryDataSize para cada fuente de datos.

Tabla 55. Valores mínimo, máximo y por omisión de queryDataSize

Fuente de datos	Versión del producto	Por omisión	Mínimo	Máximo	Valores válidos
DB2 Database para Linux, UNIX y Windows	Todos	32767	4096	262143	4096-32767, 98303, 131071, 163839, 196607, 229375, 262143[1]
IBM Informix	Todos	32767	4096	10485760	4096-10485760
DB2 para i	V5R4	32767	4096	65535	4096-65535
DB2 para i	V6R1	32767	4096	262143	4096-65535, 98303, 131071, 163839, 196607, 229375, 262143[1]
DB2 para z/OS	Versión 8 (IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4)	32767	32767	32767	32767
DB2 para z/OS	Versión 9 (IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4)	32767	32767	65535	32767, 65535
DB2 para z/OS	Versión 10 (IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4)	32767	32767	262143	32767, 65535, 98303, 131071, 163839, 196607, 229375, 262143[1]
DB2 para z/OS	Versión 10 (IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2)	32767	32767	1048575	32767, 65535, 98303, 131071, 163839, 196607, 229375, 262143, 294911, 327679, 360447, 393215, 425983, 458751, 491519, 524287, 557055, 589823, 622591, 655359, 688127, 720895, 753663, 786431, 819199, 851967, 884735, 917503, 950271, 983039, 1015807, 1048575[1]

Nota:

1. Si especifica un valor comprendido entre el valor mínimo y máximo que no sea válido, IBM Data Server Driver para JDBC y SQLJ establece queryDataSize en el valor válido más próximo.

queryTimeoutInterruptProcessingMode

Especifica qué sucede cuando termina el intervalo de tiempo de espera de consulta de un objeto Statement. Los valores válidos son:

DB2BaseDataSource.-
INTERRUPT_PROCESSING_MODE_STATEMENT_CANCEL (1)

Especifica que, cuando termina el intervalo de tiempo de espera de consulta para un objeto Statement, IBM Data Server Driver para JDBC y SQLJ cancela la sentencia de SQL que se está ejecutando en ese momento, siempre y cuando el servidor de datos soporte la

interrupción de sentencias de SQL. Si el servidor de datos no soporta la interrupción de sentencias de SQL, el controlador emite una Exception que indica que no se da soporte a la función.

DB2BaseDataSource.-
INTERRUPT_PROCESSING_MODE_STATEMENT_CANCEL es el valor por omisión. Para las conexiones con servidores de datos DB2 para z/OS, este valor está permitido, pero no se utiliza. En su lugar, se utiliza el valor 2.

DB2BaseDataSource.INTERRUPT_PROCESSING_MODE_CLOSE_SOCKET (2)
Especifica que el socket subyacente se descarta, y que la conexión se cierra cuando ha caducado el intervalo de tiempo de espera excedido de consulta para un objeto Statement.

Cuando el objeto Statement excede el tiempo de espera y no hay redireccionamientos automáticos de cliente ni afinidades de cliente configurados, se emite una DisconnectException con el código de error de SQL -4499. Todas las operaciones posteriores en el objeto Statement, o cualquier otro objeto Statement creado a partir de la misma conexión, reciben una Exception que indica que la conexión está cerrada. Después de que un objeto Statement excede el tiempo de espera, la aplicación debe establecer una nueva conexión para poder ejecutar una nueva transacción.

Si hay redireccionamientos automáticos de cliente o afinidades de cliente configurados, IBM Data Server Driver para JDBC y SQLJ intenta volver a establecer una conexión de acuerdo con el mecanismo de redireccionamiento vigente. Si se restablece correctamente una conexión nueva, el controlador devuelve un código de error de SQL -4498 o -30108, en lugar de -4499. Sin embargo, el controlador no ejecuta de nuevo las sentencias de SQL que han excedido el tiempo de espera, aunque enableSeamlessFailover se haya establecido en DB2BaseDataSource.YES (1).

Para las conexiones con servidores de datos DB2 para z/OS, IBM Data Server Driver para JDBC y SQLJ siempre utiliza este valor, con independencia del valor que se haya especificado.

resultSetHoldability
Especifica si los cursores permanecen abiertos después de una operación de confirmación. El tipo de datos de esta propiedad es int. Los valores válidos son:

DB2BaseDataSource.HOLD_CURSORS_OVER_COMMIT (1)
Dejar los cursores abiertos después de una operación de confirmación.

Este valor no es válido para conexiones que forman parte de una transacción distribuida (XA).

DB2BaseDataSource.CLOSE_CURSORS_AT_COMMIT (2)
Cerrar los cursores después de una operación de confirmación.

DB2BaseDataSource.NOT_SET (0)
Éste es el valor por omisión. El comportamiento es:

- Para las conexiones que forman parte de transacciones (XA) distribuidas, los cursores se cierran tras una operación de confirmación.
- Para las conexiones que no forman parte de una transacción distribuida:

- Para las conexiones con todas las versiones de DB2 para z/OS, DB2 Database para Linux, UNIX y Windows o DB2 para servidores i, o con Cloudscape Versión 8.1 o servidores posteriores, los cursores continúan abiertos tras una operación de confirmación.
- Para las conexiones con todas las versiones de IBM Informix, o con versiones de Cloudscape anteriores a la Versión 8.1, los cursores se cierran tras una operación de confirmación.

retrieveMessagesFromServerOnGetMessage

Especifica si las llamadas a SQLException.getMessage o SQLWarning.getMessage de JDBC hacen que IBM Data Server Driver para JDBC y SQLJ invoque un procedimiento almacenado de DB2 para z/OS que recupera el texto del mensaje para el error. El tipo de datos de esta propiedad es booleano. El valor por omisión es `false`, que significa que el texto completo del mensaje no se devuelve al cliente.

Por ejemplo, si retrieveMessagesFromServerOnGetMessage está establecido en `true`, SQLException.getMessage devuelve un mensaje similar al siguiente cuando se intenta realizar una operación de SQL sobre una tabla ADMF001.NO_TABLE inexistente:

```
ADMF001.NO_TABLE IS AN UNDEFINED NAME. SQLCODE=-204,
SQLSTATE=42704, DRIVER=3.50.54
```

Si retrieveMessagesFromServerOnGetMessage está establecido en `false`, se devuelve un mensaje similar al siguiente:

```
DB2 SQL Error: SQLCODE=-204, SQLSTATE=42704, DRIVER=3.50.54
```

En lugar de establecer esta propiedad en `true`, una alternativa consiste en utilizar el método `DB2Sqlca.getMessage`, específico de IBM Data Server Driver para JDBC y SQLJ, en aplicaciones. Ambas técnicas producen una llamada de procedimiento almacenado, la cual inicia una unidad de trabajo.

retryIntervalForClientReroute

Para el redireccionamiento automático del cliente, especifica la cantidad de tiempo, en segundos, que transcurre entre los reintentos de conexión.

El tipo de datos de esta propiedad es int.

IBM Data Server Driver para JDBC y SQLJ utiliza la propiedad retryIntervalForClientReroute solamente si también está establecida la propiedad maxRetriesForClientReroute.

Si maxRetriesForClientReroute o retryIntervalForClientReroute no está establecido, IBM Data Server Driver para JDBC y SQLJ realiza reintentos durante 10 minutos.

Si el valor de enableClientAffinitiesList es DB2BaseDataSource.NO (2), el intento de conectar con el servidor primario y los servidores alternativos se contabiliza como un reintento. El controlador espera el número de segundos especificado por retryIntervalForClientReroute antes de reintentar la conexión. Si el valor de enableClientAffinitiesList es DB2BaseDataSource.YES (1), se intenta conectar con cada servidor especificado por los valores de clientRerouteAlternateServerName y clientRerouteAlternatePortNumber una vez transcurrido el número de segundos especificado por retryIntervalForClientReroute.

El valor por omisión de retryIntervalForClientReroute viene determinado de la siguiente forma:

- Si enableClientAffinitiesList es DB2BaseDataSource.YES (1), el valor por omisión es 0.
- Para la Versión 3.63, 4.13 o posterior de IBM Data Server Driver para JDBC y SQLJ:
  - Si la propiedad enableSysplexWLB se establece en `false`, o si el servidor de datos no es DB2 para z/OS, y no se ha establecido maxRetriesForClientReroute ni retryIntervalForClientReroute, la conexión se reintenta durante 10 minutos, con un tiempo de espera entre reintentos que aumenta a medida que se incrementa el tiempo transcurrido desde el primer reintento.
  - Si la propiedad enableSysplexWLB se establece en `true`, el servidor de datos es DB2 para z/OS y no se ha establecido maxRetriesForClientReroute ni retryIntervalForClientReroute, el valor por omisión es 0.
- Para las versiones de IBM Data Server Driver para JDBC y SQLJ anteriores a la 3.63 o la 4.13, si no se ha establecido maxRetriesForClientReroute ni retryIntervalForClientReroute, la conexión se reintenta durante 10 minutos, con un tiempo de espera entre reintentos que aumenta a medida que se incrementa el tiempo transcurrido desde el primer reintento.

securityMechanism
Especifica el mecanismo de seguridad de DRDA. El tipo de datos de esta propiedad es int. Los valores posibles son:

CLEAR_TEXT_PASSWORD_SECURITY (3)
ID de usuario y contraseña

USER_ONLY_SECURITY (4)
ID de usuario solamente

ENCRYPTED_PASSWORD_SECURITY (7)
ID de usuario, contraseña cifrada

ENCRYPTED_USER_AND_PASSWORD_SECURITY (9)
ID de usuario y contraseña cifrados

KERBEROS_SECURITY (11)
Kerberos. Este valor no se aplica a las conexiones con IBM Informix.

ENCRYPTED_USER_AND_DATA_SECURITY (12)
ID de usuario y datos confidenciales cifrados. Este valor es aplicable solamente a las conexiones con DB2 para z/OS.

ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY (13)
ID de usuario y contraseña cifrados, y datos confidenciales cifrados. Este valor no se aplica a las conexiones con IBM Informix.

PLUGIN_SECURITY (15)
Seguridad por plugin. Este valor es aplicable solamente a las conexiones con DB2 Database para Linux, UNIX y Windows.

ENCRYPTED_USER_ONLY_SECURITY (16)
ID de usuario cifrado. Este valor no se aplica a las conexiones con IBM Informix.

TLS_CLIENT_CERTIFICATE_SECURITY (18)
Seguridad de certificado de cliente mediante SSL. Este valor es aplicable solamente a las conexiones con DB2 para z/OS Versión 10 y posteriores.

Si se especifica esta propiedad, el mecanismo de seguridad especificado es el único mecanismo utilizado. Si el mecanismo de seguridad no está soportado por la conexión, se emite una excepción.

El valor por omisión para securityMechanism lo proporciona la propiedad de configuración db2.jcc.securityMechanism. Si tampoco se especifica la propiedad de configuración db2.jcc.securityMechanism, el valor por omisión para securityMechanism es CLEAR_TEXT_PASSWORD_SECURITY.

Si el servidor de datos no da soporte a CLEAR_TEXT_PASSWORD_SECURITY pero sí da soporte a ENCRYPTED_USER_AND_PASSWORD_SECURITY, el controlador IBM Data Server Driver para JDBC y SQLJ actualiza el mecanismo de seguridad por ENCRYPTED_USER_AND_PASSWORD_SECURITY e intenta establecer la conexión con el servidor. Cualquier otra discrepancia en el soporte del mecanismo de seguridad entre el solicitante y el servidor da como resultado un error.

Esa propiedad no se aplica a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS.

Los mecanismos de seguridad ENCRYPTED_PASSWORD_SECURITY, ENCRYPTED_USER_AND_PASSWORD_SECURITY, ENCRYPTED_USER_AND_DATA_SECURITY, ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY y ENCRYPTED_USER_ONLY_SECURITY utilizan el cifrado DRDA. El cifrado DRDA no tiene como finalidad proporcionar confidencialidad e integridad de contraseñas o datos a través de una red que no es segura, Internet. El cifrado DRDA utiliza un intercambio de claves anónimo, Diffie-Hellman, que no proporciona autentificación del servidor o del cliente. El cifrado DRDA es vulnerable ante ataques de intrusos.

sendDataAsIs
Especifica que IBM Data Server Driver para JDBC y SQLJ no convierte valores de parámetro de entrada para los tipos de datos de columna de destino. El tipo de datos de esta propiedad es booleano. El valor por omisión es `false`.

Utilice esta propiedad solamente para aplicaciones que siempre comprueban que los tipos de datos de la aplicación coinciden con los tipos de datos de las correspondientes tablas de base de datos.

serverName
Nombre de sistema principal o dirección TCP/IP de la fuente de datos. El tipo de datos de esta propiedad es String.

sslConnection
Especifica si IBM Data Server Driver para JDBC y SQLJ utiliza un socket SSL para conectar con la fuente de datos. Si sslConnection se establece en `true`, la conexión utilizará un socket SSL. Si sslConnection se establece en `false`, la conexión utilizará un socket normal.

Esta propiedad solo es aplicable al IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4.

sslTrustStoreLocation
Especifica el nombre del almacén de confianza de Java en el cliente que contiene el certificado del servidor para una conexión SSL.

IBM Data Server Driver para JDBC y SQLJ sólo utiliza esta opción si la propiedad sslConnection está establecida en `true`.

Si sslTrustStore está establecida y sslConnection está establecida en `true`, IBM Data Server Driver para JDBC y SQLJ utiliza el valor de sslTrustStoreLocation en lugar del valor en la propiedad Java javax.net.ssl.trustStore.

Esta propiedad solo es aplicable al IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4.

sslTrustStorePassword
Especifica la contraseña para el almacén de confianza de Java en el cliente que contiene el certificado del servidor para una conexión SSL.

IBM Data Server Driver para JDBC y SQLJ sólo utiliza esta opción si la propiedad sslConnection está establecida en `true`.

Si sslTrustStorePassword está establecida y sslConnection está establecida en `true`, IBM Data Server Driver para JDBC y SQLJ utiliza el valor de sslTrustStorePassword en lugar del valor en la propiedad Java javax.net.ssl.trustStorePassword.

Esta propiedad solo es aplicable al IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4.

stripTrailingZerosForDecimalNumbers
Especifica si IBM Data Server Driver para JDBC y SQLJ elimina los ceros finales cuando recupera datos de una columna DECFLOAT, DECIMAL o NUMERIC. Esta propiedad solamente es significativa si el SDK de Java pertenece a la versión 1.5 o a una versión posterior. El tipo de datos de esta propiedad es int.

Los valores posibles son:

DB2BaseDataSource.NOT_SET (0)
IBM Data Server Driver para JDBC y SQLJ no elimina los ceros finales del valor recuperado. Éste es el valor por omisión.

DB2BaseDataSource.YES (1)
IBM Data Server Driver para JDBC y SQLJ elimina los ceros finales cuando recupera un valor de una columna DECFLOAT, DECIMAL o NUMERIC como un objeto java.math.BigDecimal.

Por ejemplo, cuando el controlador recupera el valor 234,04000, devuelve el valor 234,04 a la aplicación.

DB2BaseDataSource.NO (2)
IBM Data Server Driver para JDBC y SQLJ no elimina los ceros finales del valor recuperado.

timerLevelForQueryTimeOut
Especifica el nivel en el que IBM Data Server Driver para JDBC y SQLJ crea un objeto java.util.Timer para esperar a que la ejecución de una consulta exceda el tiempo de espera. Los valores posibles son:

DB2BaseDataSource.QUERYTIMEOUT_STATEMENT_LEVEL (1)
IBM Data Server Driver para JDBC y SQLJ crea un objeto Timer para cada objeto Statement. Cuando se cierra el objeto Statement, el controlador suprime el objeto Timer. Éste es el valor por omisión.

DB2BaseDataSource.QUERYTIMEOUT_CONNECTION_LEVEL (2)
IBM Data Server Driver para JDBC y SQLJ crea un objeto Timer para cada objeto Connection. Cuando se cierra el objeto Connection, el controlador suprime el objeto Timer.

DB2BaseDataSource.QUERYTIMEOUT_DISABLED (-1)
IBM Data Server Driver para JDBC y SQLJ no crea un objeto Timer para controlar el tiempo de espera de ejecución de consultas.

timestampFormat
Especifica el formato en que se devuelve el resultado del método `ResultSet.getString` o `CallableStatement.getString` sobre una columna TIMESTAMP. El tipo de datos de timestampFormat es int.

Los valores posibles de timestampFormat son:

Constante	Valor entero	Formato
`com.ibm.db2.jcc.DB2BaseDataSource.ISO`	1	*aaaa-mm-dd-hh.mm.ss.nnnnnnnnn*[1]
`com.ibm.db2.jcc.DB2BaseDataSource.JDBC`	5	*aaaa-mm-dd hh:mm:ss.nnnnnnnnn*[1]

Nota:

1. El número de dígitos de la parte fraccionaria de la indicación de fecha y hora depende de la precisión de la columna TIMESTAMP(*p*) de la tabla fuente. Si *p*<9, se devuelven *p* dígitos. Si *p*>=9, se devuelven 9 dígitos, y los dígitos restantes se truncan.

El valor por omisión es `com.ibm.db2.jcc.DB2BaseDataSource.JDBC`.

timestampFormat sólo afecta al formato de salida.

timestampPrecisionReporting
Especifica los ceros de cola se truncan en el resultado de una llamada de Resultset.getString para un valor TIMESTAMP. El tipo de datos de esta propiedad es int. Los valores posibles son:

TIMESTAMP_JDBC_STANDARD (1)
Los ceros de cola se truncan en el resultado de una llamada de Resultset.getString para un valor TIMESTAMP. Éste es el valor por omisión.

Por ejemplo:

- Un valor TIMESTAMP de 2009-07-19-10.12.00.000000 se trunca en 2009-07-19-10.12.00.0 después de la recuperación.
- Un valor TIMESTAMP de 2009-12-01-11.30.00.100000 se trunca en 2009-12-01-11.30.00.1 después de la recuperación.

TIMESTAMP_ZERO_PADDING (2)
Los ceros de cola no se truncan en el resultado de una llamada de Resultset.getString para un valor TIMESTAMP.

traceDirectory
Especifica un directorio en el que se graba la información de rastreo. El tipo de datos de esta propiedad es `String`. Cuando se especifica traceDirectory, la información de rastreo correspondiente a varias conexiones en el mismo DataSource se graba en varios archivos.

Si se especifica traceDirectory, la conexión se guarda en un archivo denominado traceFile_*origen*_*n*.

n es la conexión número *n* correspondiente a una DataSource.

origen indica el origen del grabador de anotaciones cronológicas que se está utilizando. Los valores posibles de *origen* son:

cpds Grabador de anotaciones cronológicas para un objeto DB2ConnectionPoolDataSource.

driver Grabador de anotaciones cronológicas para un objeto DB2Driver.

global Grabador de anotaciones cronológicas para un objeto DB2TraceManager.

sds Grabador de anotaciones cronológicas para un objeto DB2SimpleDataSource.

xads Grabador de anotaciones cronológicas para un objeto DB2XADataSource.

Si también se especifica la propiedad traceFile, no se utiliza el valor traceDirectory.

traceFile
Especifica el nombre del archivo en donde IBM Data Server Driver para JDBC y SQLJ escribe información de rastreo. El tipo de datos de esta propiedad es `String`. La propiedad traceFile es una alternativa al uso de la propiedad logWriter para encaminar la corriente de datos de rastreo de salida hacia un archivo.

traceFileAppend
Especifica si deben añadir o sobrescribir datos en el archivo especificado por la propiedad traceFile. El tipo de datos de esta propiedad es booleano. El valor por omisión es `false`, que significa que se sobrescribe el archivo especificado por la propiedad traceFile.

traceLevel
Especifica qué se debe rastrear. El tipo de datos de esta propiedad es int.

Puede especificar uno o más de los valores de rastreo siguientes con la propiedad traceLevel:
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_NONE` (X'00')
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_CONNECTION_CALLS` (X'01')
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_STATEMENT_CALLS` (X'02')
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_RESULT_SET_CALLS` (X'04')
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DRIVER_CONFIGURATION` (X'10')
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_CONNECTS` (X'20')
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DRDA_FLOWS` (X'40')
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_RESULT_SET_META_DATA` (X'80')
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_PARAMETER_META_DATA` (X'100')
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DIAGNOSTICS` (X'200')
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_SQLJ` (X'400')
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_XA_CALLS (sólo IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 para DB2 Database para Linux, UNIX y Windows)` (X'800')
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_META_CALLS` (X'2000')
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DATASOURCE_CALLS` (X'4000')
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_LARGE_OBJECT_CALLS` (X'8000')
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_SYSTEM_MONITOR` (X'20000')
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_TRACEPOINTS` (X'40000')
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL` (X'FFFFFFFF')

Para especificar más de un valor de rastreo, utilice una de estas técnicas:
- Utilice operadores OR (|) de bits con dos o más valores de rastreo. Por ejemplo, para rastrear flujos de DRDA y llamadas de conexión, especifique este valor para traceLevel:

  ```
  TRACE_DRDA_FLOWS|TRACE_CONNECTION_CALLS
  ```

- Utilice un operador de complemento a nivel de bit (~) con un valor de rastreo para especificar todos los rastreos excepto uno determinado. Por ejemplo, para rastrear todo excepto los flujos de DRDA, especifique este valor para traceLevel:

  ```
  ~TRACE_DRDA_FLOWS
  ```

traceFileCount
Especifica el número máximo de archivos de rastreo para el rastreo circular. IBM Data Server Driver para JDBC y SQLJ sólo utiliza esta propiedad cuando traceOption se establece en DB2BaseDataSource.TRACE_OPTION_CIRCULAR (1). El tipo de datos de esta propiedad es int. El valor por omisión es 2.

traceFileSize
Especifica el tamaño máximo de cada archivo de rastreo para el rastreo circular. IBM Data Server Driver para JDBC y SQLJ sólo utiliza esta propiedad cuando traceOption se establece en DB2BaseDataSource.TRACE_OPTION_CIRCULAR (1). El tipo de datos de esta propiedad es int. El valor por omisión es 10485760 (10 MB).

useJDBC41DefinitionForGetColumns
Especifica si el método DatabaseMetaData.getColumns devuelve un conjunto de resultados con una columna con el nombre SCOPE_CATALOG o SCOPE_CATLOG. Los valores posibles son:

DB2BaseDataSource.NOT_SET (0)
Especifica que, para la versión 4.13 o posteriores de IBM Data Server Driver para JDBC y SQLJ, el conjunto de resultados de DatabaseMetaData.getColumns contiene una columna denominada SCOPE_CATALOG. Para la versión 4.12 o anteriores de IBM Data Server Driver para JDBC y SQLJ, esa columna se denomina SCOPE_CATLOG.

DB2BaseDataSource.YES (1)
Especifica que, para la versión 4.13 o posteriores de IBM Data Server Driver para JDBC y SQLJ, el conjunto de resultados de DatabaseMetaData.getColumns contiene una columna denominada SCOPE_CATALOG. Para la versión 4.12 o anteriores de IBM Data Server Driver para JDBC y SQLJ, esa columna se denomina SCOPE_CATLOG.

DB2BaseDataSource.NO (2)
Especifica que, para todas las versiones de IBM Data Server Driver para JDBC y SQLJ, el conjunto de resultados de DatabaseMetaData.getColumns contiene una columna denominada SCOPE_CATLOG.

traceOption
Especifica el modo como se recopilan los datos de rastreo. El tipo de datos de esta propiedad es int. Los valores posibles son:

DB2BaseDataSource.NOT_SET (0)
Especifica que se genera un solo archivo de rastreo y que no existe ningún límite para el tamaño del archivo. Éste es el valor por omisión.

Si el valor de traceOption es NOT_SET, se pasan por alto las propiedades traceFileSize y traceFileCount.

DB2BaseDataSource.TRACE_OPTION_CIRCULAR (1)
Especifica que IBM Data Server Driver para JDBC y SQLJ realiza un rastreo circular. El rastreo circular se realiza de la forma siguiente:

1. Cuando una aplicación graba su primer registro de rastreo, el controlador crea un archivo.
2. El controlador graba los datos de rastreo en el archivo.
3. Cuando el tamaño del archivo es igual al valor de la propiedad traceFileSize, el controlador crea otro archivo.
4. El controlador repite los pasos 2 y 3 hasta que el número de archivos en los que se han grabado datos es igual al valor de la propiedad traceFileCount.
5. El controlador graba datos en el primer archivo de rastreo y sobrescribe los datos existentes.
6. El controlador repite los pasos de 3 a 5 hasta que la aplicación finaliza.

Los nombres de los archivos de rastreo son los nombres de archivo que la propiedad traceFile o traceDirectory determina, con .1 para el primer archivo, .2 para el segundo archivo y así sucesivamente.

user
ID de usuario que se utilizará para establecer conexiones. El tipo de datos de esta propiedad es String. Cuando utiliza la interfaz DataSource para establecer una conexión, puede alterar temporalmente el valor de esta propiedad invocando esta modalidad del método DataSource.getConnection:

```
getConnection(usuario, contraseña);
```

xaNetworkOptimization
Especifica si la optimización de la red XA está habilitada para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4. Puede que tenga que inhabilitar la optimización de la red XA en un entorno en el que se emitan XA Start y XA End desde un proceso Java, y se emitan XA Prepare y XA Commit desde otro proceso Java. Con la optimización de la red XA, XA Prepare puede acceder a la fuente de datos antes del XA End, lo cual produce un error XAER_PROTO. Para evitar el error XAER_PROTO, inhabilite la optimización de la red XA.

El valor por omisión es `true`, que significa que la optimización de la red XA está habilitada. Si el valor de xaNetworkOptimization es `false`, lo que significa que la optimización de la red XA está inhabilitada, el controlador cerrará los cursores abiertos al realizar la operación XA End.

xaNetworkOptimization se puede establecer en un objeto DataSource o en el parámetro *url* en una llamada getConnection. El valor de xaNetworkOptimization no se puede modificar después de obtener una conexión.

Propiedades comunes de IBM Data Server Driver para JDBC y SQLJ para servidores DB2

Algunas de las propiedades de IBM Data Server Driver para JDBC y SQLJ son aplicables solamente a DB2 para z/OS y DB2 Database para Linux, UNIX y Windows.

A menos que se indique otra cosa, todas las propiedades están contenidas en com.ibm.db2.jcc.DB2BaseDataSource.

Estas propiedades son las siguientes:

alternateGroupDatabaseName
Especifica los nombres de base de datos de los grupos alternativos con los que

se puede conectar una aplicación. El tipo de datos de esta propiedad es String. En el caso de una conexión con un servidor de datos DB2 para z/OS, este valor es el nombre de ubicación de un grupo de compartimiento de datos. En el caso de una conexión con un servidor de datos DB2 Database para Linux, UNIX y Windows, cada uno de estos valores es el nombre de base de datos de una instancia DB2 pureScale. Si se especifica más de un nombre de base de datos, los nombres de base de datos deben estar separados por comas.

En el caso de conexiones con DB2 para z/OS, sólo se puede especificar un valor.

alternateGroupPortNumber
Especifica los números de puerto de los grupos alternativos con los que se puede conectar una aplicación. El tipo de datos de esta propiedad es String. En el caso de una conexión con un servidor de datos DB2 para z/OS, este valor es el número de puerto del servidor TCP/IP asignado al grupo de compartimiento de datos. En el caso de una conexión con un servidor de datos DB2 Database para Linux, UNIX y Windows, cada uno de estos valores es el número de puerto del servidor TCP/IP asignado a una instancia DB2 pureScale. Si se especifica más de un número de puerto, los números de puerto deben ir separados por comas.

En el caso de conexiones con DB2 para z/OS, sólo se puede especificar un valor.

alternateGroupServerName
Especifica los nombres de sistema principal de los grupos alternativos con los que se puede conectar una aplicación. El tipo de datos de esta propiedad es String. El tipo de datos de esta propiedad es String. En el caso de una conexión con un servidor de datos DB2 para z/OS, este valor es el nombre de dominio o la dirección IP asignados al grupo de compartimiento de datos. En el caso de una conexión con un servidor de datos DB2 Database para Linux, UNIX y Windows, cada uno de estos valores es el nombre de dominio o dirección IP asociados a una instancia DB2 pureScale. Si se especifica más de un nombre de sistema principal, los nombres de sistema principal deben ir separados por comas.

En el caso de conexiones con DB2 para z/OS, sólo se puede especificar un valor.

clientAccountingInformation
Especifica información contable correspondiente al cliente actual de la conexión. Esta información se utiliza con fines de contabilidad de clientes. Este valor puede cambiar durante una conexión. El tipo de datos de esta propiedad es String. La longitud máxima es de 255 bytes. Una serie vacía Java ("") es válida para este valor, pero un valor `null` de Java no es válido.

clientApplicationInformation
Especifica el nombre de aplicación o transacción de la aplicación del usuario final. Utilice esta propiedad para proporcionar la identidad del usuario final cliente con fines de contabilidad y supervisión. Este valor puede cambiar durante una conexión. El tipo de datos de esta propiedad es String. Para un servidor DB2 para z/OS, la longitud máxima es 32 bytes. Para un servidor DB2 Database para Linux, UNIX y Windows, la longitud máxima es de 255 bytes. Una serie vacía Java ("") es válida para este valor, pero un valor `null` de Java no es válido.

clientProgramId
Especifica un valor para ID del programa cliente que sirve para identificar al

usuario final. El tipo de datos de esta propiedad es String, y la longitud es 80 bytes. Si el valor del ID de programa es menor que 80 bytes, el valor se debe rellenar con blancos.

clientProgramName
Especifica el ID de aplicación que se utilizar mientras dure la conexión física para un cliente. El valor de esta propiedad se convierte en el ID de correlación en un servidor DB2 para z/OS. Los administradores de base de datos pueden utilizar esta propiedad para correlacionar tareas en un servidor DB2 para z/OS con aplicaciones de cliente. El tipo de datos de esta propiedad es String. La longitud máxima es de 12 bytes. Si este valor es `null`, IBM Data Server Driver para JDBC y SQLJ proporcionará un valor db2jcc*nombre-hebra*.

Esta propiedad sólo se aplica a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4.

concurrentAccessResolution
Especifica si IBM Data Server Driver para JDBC y SQLJ solicita que una transacción de lectura pueda acceder a una imagen confirmada y consistente de filas que están bloqueadas por transacciones de grabación de forma incompatible, si la fuente de datos soporta el acceso a los datos confirmados actualmente y si el nivel de aislamiento de la aplicación es estabilidad de cursor (CS) o estabilidad de lectura (RS). Esta opción tiene el mismo efecto que la opción de vinculación CONCURRENTACCESSRESOLUTION de DB2. Los valores posibles son:

**DB2BaseDataSource.-
CONCURRENTACCESS_USE_CURRENTLY_COMMITTED (1)**
IBM Data Server Driver para JDBC y SQLJ solicita lo siguiente:

- Las transacciones de lectura deben acceder a los datos confirmados actualmente cuando se están actualizando o suprimiendo los datos.
- Las transacciones de lectura deben omitir las filas que se están insertando.

DB2BaseDataSource.CONCURRENTACCESS_WAIT_FOR_OUTCOME (2)
IBM Data Server Driver para JDBC y SQLJ solicita lo siguiente:

- Las transacciones de lectura deben esperar a una operación de confirmación o retrotracción cuando encuentran datos que se están actualizando o suprimiendo.
- Las transacciones de lectura no deben omitir las filas que se están insertando.

DB2BaseDataSource.CONCURRENTACCESS_NOT_SET (0)
Habilita el comportamiento por omisión del servidor de datos para las transacciones de lectura cuando se produce una contención de bloqueo. Éste es el valor por omisión.

currentDegree
Especifica el grado de paralelismo para la ejecución de consultas que se preparan dinámicamente. El tipo de datos de esta propiedad es String. El valor de currentDegree se utiliza para establecer el registro especial CURRENT DEGREE en la fuente de datos. Si currentDegree no está definido, no se pasa ningún valor a la fuente de datos.

currentExplainMode
Especifica el valor del registro especial CURRENT EXPLAIN MODE. El registro especial CURRENT EXPLAIN MODE habilita e inhabilita el recurso Explain. El tipo de datos de esta propiedad es String. La longitud máxima es

de 254 bytes. Esta propiedad solamente es aplicable a conexiones con fuentes de datos que son compatibles con el registro especial CURRENT EXPLAIN MODE.

currentFunctionPath
Especifica la vía de SQL que se utiliza para resolver nombres de tipos de datos y de funciones no calificados en las sentencias de SQL que están contenidas en programas de JDBC. El tipo de datos de esta propiedad es String. Para un servidor DB2 Database para Linux, UNIX y Windows, la longitud máxima es de 254 bytes. Para un servidor DB2 para z/OS, la longitud máxima es de 2048 bytes. El valor es una lista de nombres de esquema separados por comas. Esos nombres pueden ser identificadores ordinarios o delimitados.

currentMaintainedTableTypesForOptimization
Especifica un valor que identifica los tipos de objetos que se pueden tener en cuenta cuando la fuente de datos optimiza el proceso de consultas de SQL dinámico. Este registro contiene una palabra clave que representa tipos de tabla. El tipo de datos de esta propiedad es String.

Los valores posibles de currentMaintainedTableTypesForOptimization son:

ALL
Indica que se tendrán en cuenta todas las tablas de consulta materializada.

NONE
Indica que no se tendrá en cuenta ninguna tabla de consulta materializada.

SYSTEM
Indica que sólo se tendrán en cuenta las tablas de consulta materializada de renovación diferida mantenidas por el sistema.

USER
Indica que sólo se tendrán en cuenta las tablas de consulta materializada de renovación diferida mantenidas por el usuario.

currentPackagePath
Especifica una lista separada por comas de colecciones contenidas en el servidor. El servidor de bases de datos busca paquetes de JDBC y SQLJ en estas colecciones.

Las reglas de precedencia para las propiedades currentPackagePath y currentPackageSet siguen las reglas de precedencia correspondientes a los registros especiales CURRENT PACKAGESET y CURRENT PACKAGE.

currentPackageSet
Especifica el ID de colección para buscar paquetes JDBC y SQLJ. El tipo de datos de esta propiedad es String. El valor por omisión es NULLID. Si currentPackageSet está definido, su valor tiene prioridad sobre el valor de jdbcCollection.

Puede instalar varias instancias de IBM Data Server Driver para JDBC y SQLJ en un servidor de bases de datos ejecutando varias veces el programa de utilidad DB2Binder. El programa de utilidad DB2binder incluye la opción -collection, que permite que el instalador especifique el ID de colección para cada instancia de IBM Data Server Driver para JDBC y SQLJ. Para seleccionar una instancia de IBM Data Server Driver para JDBC y SQLJ para una conexión, especifique un valor de currentPackageSet que coincida con el ID de colección para una de las instancias de IBM Data Server Driver para JDBC y SQLJ.

Las reglas de precedencia para las propiedades currentPackagePath y currentPackageSet siguen las reglas de precedencia correspondientes a los registros especiales CURRENT PACKAGESET y CURRENT PACKAGE.

currentRefreshAge
Especifica un valor de duración de indicación de fecha y hora máxima desde que la sentencia REFRESH TABLE se ha procesado en una tabla de consultas materializada REFRESH DEFERRED mantenida por el sistema, como por ejemplo en el caso en el que la tabla de consultas materializada se utiliza para optimizar el proceso de una consulta. Esta propiedad afecta a la coincidencia de antememoria de sentencias dinámicas. El tipo de datos de esta propiedad es long.

currentSchema
Especifica el nombre de esquema por omisión que se utiliza para calificar objetos de base de datos no calificados en sentencias de SQL preparadas dinámicamente. El valor de esta propiedad establece el valor del registro especial CURRENT SCHEMA en el servidor de bases de datos. El nombre de esquema es sensible a las mayúsculas y minúsculas y debe especificarse en caracteres en mayúsculas.

cursorSensitivity
Especifica si el valor de `java.sql.ResultSet.TYPE_SCROLL_SENSITIVE` para un ResultSet se correlaciona con el atributo SENSITIVE DYNAMIC, el atributo SENSITIVE STATIC, o el atributo ASENSITIVE para el cursor de base de datos subyacente. El tipo de datos de esta propiedad es int. Los valores posibles son `TYPE_SCROLL_SENSITIVE_STATIC` (0), `TYPE_SCROLL_SENSITIVE_DYNAMIC` (1), o `TYPE_SCROLL_ASENSITIVE` (2). El valor por omisión es `TYPE_SCROLL_SENSITIVE_STATIC`.

Si la fuente de datos no es compatible con cursores desplazables dinámicos sensibles, y se especifica `TYPE_SCROLL_SENSITIVE_DYNAMIC`, el controlador JDBC obtiene un aviso y establece la sensibilidad en SENSITIVE STATIC. Para servidores de bases de datos DB2 para i, que no dan soporte a los cursores estáticos sensibles, `java.sql.ResultSet.TYPE_SCROLL_SENSITIVE` siempre se correlaciona con SENSITIVE DYNAMIC.

dateFormat
Especifica:

- El formato en que se debe especificar el argumento String del método PreparedStatement.setString sobre una columna DATE.
- El formato en que se devuelve el resultado del método ResultSet.getString o CallableStatement.getString sobre una columna DATE.

El tipo de datos de dateFormat es int.

Los valores posibles de dateFormat son:

Constante	Valor entero	Formato
`com.ibm.db2.jcc.DB2BaseDataSource.ISO`	1	*aaaa-mm-dd*
`com.ibm.db2.jcc.DB2BaseDataSource.USA`	2	*mm/dd/aaaa*
`com.ibm.db2.jcc.DB2BaseDataSource.EUR`	3	*dd.mm.aaaa*
`com.ibm.db2.jcc.DB2BaseDataSource.JIS`	4	*aaaa-mm-dd*

El valor por omisión es `com.ibm.db2.jcc.DB2BaseDataSource.ISO`.

decimalRoundingMode
Especifica la modalidad de redondeo para la asignación de variables de coma flotante decimal o columnas DECFLOAT en servidores de datos DB2 para z/OS o DB2 Database para Linux, UNIX y Windows.

Los valores posibles son:

DB2BaseDataSource.ROUND_DOWN (1)
Redondea el valor hacia 0 (truncamiento). Los dígitos descartados no se tienen en cuenta.

DB2BaseDataSource.ROUND_CEILING (2)
Redondea el valor hacia infinito positivo. Si todos los dígitos descartados son ceros, o si el signo es negativo, el resultado permanece inalterado salvo por la eliminación de los dígitos descartados. En otro caso, el coeficiente del resultado se incrementa en 1.

DB2BaseDataSource.ROUND_HALF_EVEN (3)
Redondea el valor hasta el valor más cercano; si los valores son equidistantes, redondea el valor de forma que el dígito final sea par. Si los dígitos descartados representan un valor mayor que la mitad (0,5) del valor de un dígito en la posición izquierda siguiente, el coeficiente del resultado se incrementa en 1. Si éstos son inferiores a 0,5 el coeficiente del resultado no se ajustará (es decir, que los dígitos descartados se ignorarán). En otro caso, el coeficiente del resultado permanece inalterado si su dígito más a la derecha es par, o se incrementa en 1 si su dígito más a la derecha es impar (para convertirlo en un dígito par).

DB2BaseDataSource.ROUND_HALF_UP (4)
Redondea el valor hasta el valor más cercano; si los valores son equidistantes, redondea el valor por exceso. Si los dígitos descartados representan un valor mayor o igual que la mitad (0,5) del valor del dígito en la posición izquierda siguiente, el coeficiente del resultado se incrementa en 1. En otro caso, los dígitos descartados no se tienen en cuenta.

DB2BaseDataSource.ROUND_FLOOR (6)
Redondea el valor hacia infinito negativo. Si todos los dígitos descartados son ceros, o si el signo es positivo, el resultado permanece inalterado salvo por la eliminación de los dígitos descartados. En otro caso, el signo es negativo y el coeficiente del resultado se incrementa en 1.

DB2BaseDataSource.ROUND_UNSET (-2147483647)
No se ha establecido explícitamente ninguna modalidad de redondeo. IBM Data Server Driver para JDBC y SQLJ no utiliza la propiedad decimalRoundingMode para establecer la modalidad de redondeo en el servidor de datos. La modalidad de redondeo es ROUND_HALF_EVEN.

Si se establece de forma explícita el valor de decimalRoundingMode, ese valor actualiza el valor del registro especial CURRENT DECFLOAT ROUNDING MODE en un servidor de datos DB2 para z/OS.

Si se establece de forma explícita el valor de decimalRoundingMode, ese valor no actualiza el valor del registro especial CURRENT DECFLOAT ROUNDING MODE en un servidor de datos DB2 Database para Linux, UNIX y Windows. Si el valor en el que ha establecido decimalRoundingMode no es igual que el valor del registro especial CURRENT DECFLOAT ROUNDING MODE, se genera una Exception. Para cambiar el valor del servidor de datos, tiene que establecer ese valor con el parámetro de configuración de base de datos decflt_rounding.

decimalRoundingMode no afecta las asignaciones de valores decimales. IBM Data Server Driver para JDBC y SQLJ siempre redondea a la baja los valores decimales.

enableAlternateGroupSeamlessACR
Especifica si la migración tras error a un grupo alternativo es sin fisuras o con fisuras. El tipo de datos de esta propiedad es booleano. Los valores posibles son:

false La migración tras error no es sin fisuras. `false` es el valor por omisión.

Con el comportamiento con fisuras, si una aplicación que está conectada actualmente a un grupo primario está ejecutando una transacción y todo el grupo primario se desactiva, IBM Data Server Driver para JDBC y SQLJ transfiere el control al grupo alternativo. Si la migración tras error es satisfactoria, el controlador emite una SQLException con el código de error de SQL -30108.

true La migración tras error es sin fisuras.

Con el comportamiento sin fisuras, si una aplicación que está conectada actualmente a un grupo primario está ejecutando una transacción y todo el grupo primario se desactiva, IBM Data Server Driver para JDBC y SQLJ transfiere el control al grupo alternativo. Si la transacción es admisible para la migración tras error sin fisuras, la conexión se reintenta. Si la conexión es satisfactoria, no se emite ninguna SQLException.

En el caso de conexiones con DB2 para z/OS, sólo se puede especificar un valor.

enableExtendedDescribe
Especifica si IBM Data Server Driver para JDBC y SQLJ ha de solicitar información descriptiva ampliada del servidor de datos cuando prepare una sentencia.

La información descriptiva ampliada proporciona:

- Información descriptiva adicional para un cursor o un conjunto de resultados
- Información que indique si una columna:
  - Puede actualizarse
  - Es una clave primaria o un miembro de clave candidata preferido
  - Es una expresión o una tabla de columna
  - Es una columna generada o una columna de tabla
- El nombre de vista o tabla totalmente calificado
- El nombre de columna totalmente calificado

Los valores posibles son:

DB2BaseDataSource.NOT_SET (0)
IBM Data Server Driver para JDBC y SQLJ solicita información descriptiva ampliada. Éste es el valor por omisión.

DB2BaseDataSource.YES (1)
IBM Data Server Driver para JDBC y SQLJ solicita información descriptiva ampliada.

DB2BaseDataSource.NO (2)
IBM Data Server Driver para JDBC y SQLJ no solicita información descriptiva ampliada.

El establecimiento de enableExtendedDescribe en `DB2BaseDataSource.NO` puede aportar beneficios de rendimiento, pues evita el proceso adicional que el controlador debe realizar para

proporcionar la información adicional. Sin embargo, si especifica esta opción, algunos métodos generan una excepción o devuelven resultados no esperados. En la tabla siguiente se muestra el comportamiento de los métodos cuando enableExtendedDescribe se ha establecido en `DB2BaseDataSource.NO`.

Método	Resultado cuando la descripción ampliada está desactivada
Connection.findAutoGeneratedKeysColumn	Devuelve una matriz de series vacías ("")
DB2ResultSetMetaData.getDBTemporalColumnType	Devuelve -1
ResultSet.getMetaData en el objeto ResultSet que PreparedStatement.getGeneratedKeys devuelve	Devuelve NULL
ResultSet.insertRow, ResultSet.deleteRow, ResultSet.updateRow	SQLException con el código de error -4474, SQLSTATE 42808 (columna no actualizable)
Métodos ResultSet.updateXXX	SQLException con el código de error -4474, SQLSTATE 42808 (columna no actualizable)
ResultSetMetaData.getTableName, ResultSetMetaData.getSchemaName, ResultSetMetaData.getColumnName	Devuelve una serie vacía ("")
ResultSetMetaData.isAutoIncrement	Devuelve `false`

enableExtendedIndicators
Especifica si el soporte para los indicadores ampliados está habilitado en IBM Data Server Driver para JDBC y SQLJ. Los valores posibles son:

DB2BaseDataSource.YES (1)
El soporte para los indicadores ampliados está habilitado en IBM Data Server Driver para JDBC y SQLJ.

DB2BaseDataSource.NO (2)
El soporte para los indicadores ampliados está inhabilitado en IBM Data Server Driver para JDBC y SQLJ.

DB2BaseDataSource.NOT_SET (0)
El soporte para los indicadores ampliados está habilitado en IBM Data Server Driver para JDBC y SQLJ. Éste es el valor por omisión.

enableRowsetSupport
Especifica si el controlador IBM Data Server Driver para JDBC y SQLJ utiliza operaciones FETCH de varias filas para cursores de solo avance o cursores desplazables si el servidor de datos es compatible con la operación FETCH de varias filas. El tipo de datos de esta propiedad es int.

Para las conexiones con DB2 para z/OS, cuando se ha establecido enableRowsetSupport, su valor altera temporalmente el valor de la propiedad useRowsetCursor.

Los valores posibles son:

DB2BaseDataSource.YES (1)
Especifica que:

- Para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 con DB2 para z/OS, se utiliza la operación FETCH de varias filas para cursores desplazables y cursores de solo avance si el servidor de datos da soporte a la operación FETCH de varias filas.
- Para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4, o IBM Data Server Driver para JDBC y SQLJ con conectividad

de tipo 2 con DB2 Database para Linux, UNIX y Windows, se utiliza la operación FETCH de varias filas para cursores desplazables si el servidor de datos da soporte a la operación FETCH de varias filas.

DB2BaseDataSource.NO (2)
Especifica que no se utiliza la operación FETCH de varia filas.

DB2BaseDataSource.NOT_SET (0)
Especifica que si la propiedad enableRowsetSupport no se ha establecido:

- Para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 con DB2 para z/OS, la operación FETCH de varias filas no se utiliza.
- Para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 con DB2 para z/OS, se utiliza la operación FETCH de varias filas si useRowsetCursor se establece en `true`.
- En el caso de conexiones con DB2 Database para Linux, UNIX y Windows, la operación FETCH de varias filas se utiliza para cursores desplazables, si el servidor de datos da soporte a operaciones FETCH de varias filas.

Para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 con DB2 para z/OS, la operación FETCH de varias filas no es compatible con la modalidad continua progresiva. Por tanto, si se utiliza la modalidad continua progresiva para una operación FETCH, la operación FETCH de varias filas no se utiliza.

`encryptionAlgorithm`
Especifica si el controlador IBM Data Server Driver para JDBC y SQLJ utiliza cifrado DES de 56 bits (débil) o cifrado AES de 256 bits (fuerte). El tipo de datos de esta propiedad es int. Los valores posibles son:

1 El controlador utiliza cifrado DES de 56 bits.

Este valor es el valor por omisión, a menos que la propiedad de configuración db2.jcc.encryptionAlgorithm proporcione un valor por omisión distinto.

2 El controlador utiliza cifrado AES de 256 bits, si este cifrado es compatible con el servidor de bases de datos. El cifrado EAS de 256 bits solo está disponible para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4.

Para el cifrado AES, necesita un archivo de política no restringida para JCE. Para IBM SDK para Java, el archivo está disponible en la ubicación siguiente:

```
https://www14.software.ibm.com/webapp/iwm/web/preLogin.do?source=jcesdk
```

El SDK para Java de Oracle podría soportar el cifrado de AES, pero no el cifrado DES. Si se utiliza el cifrado de AES con el SDK para Java de Oracle, es necesario tener instalado el archivo JCE Unlimited Strength Jurisdiction Policy File. Este archivo está disponible en Oracle. Si no se encuentra el archivo JCE Unlimited Strength Jurisdiction Policy File, se emite una java.security.InvalidKeyException.

encryptionAlgorithm solo puede especificarse si el valor securityMechanism o db2.jcc.securityMechanism es `ENCRYPTED_PASSWORD_SECURITY` (7) o `ENCRYPTED_USER_AND_PASSWORD_SECURITY` (9).

fullyMaterializeInputStreams
Indica si las corrientes de datos se materializan totalmente antes de ser enviadas a una fuente de datos. El tipo de datos de esta propiedad es booleano. El valor por omisión es `false`.

Si el valor de `fullyMaterializeInputStreams` es `true`, ello significará que el controlador JDBC habrá materializado completamente las corrientes antes de enviarlas al servidor.

gssCredential
Para una fuente de datos que utilice la seguridad Kerberos, especifica una credencial delegada que se pasa desde otro principal. El tipo de datos de esta propiedad es org.ietf.jgss.GSSCredential. Las credenciales delegadas se utilizan en entornos de varios niveles, como cuando un cliente se conecta a WebSphere Application Server, y éste, a su vez, se conecta a la fuente de datos. El valor de esta propiedad se obtiene del cliente, invocando el método GSSContext.getDelegCred. GSSContext forma parte de la API GSS (Generic Security Service) de IBM Java. Si define esta propiedad, debe también definir las propiedades Mechanism y KerberosServerPrincipal.

Esta propiedad solo es aplicable al IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4.

Para obtener más información sobre la utilización de la seguridad Kerberos con IBM Data Server Driver para JDBC y SQLJ, consulte "Utilización de la seguridad Kerberos bajo IBM Data Server Driver para JDBC y SQLJ".

kerberosServerPrincipal
Para una fuente de datos que utilice la seguridad Kerberos, especifica el nombre que se utiliza para la fuente de datos cuando se registra con el Centro de distribución de claves (KCD) de Kerberos. El tipo de datos de esta propiedad es String.

Esta propiedad solo es aplicable al IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4.

pdqProperties
Especifica propiedades que controlan la interacción entre IBM Data Server Driver para JDBC y SQLJ y la característica de optimización de cliente de pureQuery.

El tipo de datos de esta propiedad es String.

Defina la propiedad pdqProperties **solamente** si está utilizando la característica de optimización del cliente de pureQuery. Consulte el Centro de información de Integrated Data Management para obtener información sobre los valores válidos de pdqProperties.

readOnly
Especifica si la conexión es de solo lectura. El tipo de datos de esta propiedad es booleano. El valor por omisión es `false`.

resultSetHoldabilityForCatalogQueries
Especifica si los cursores de consultas que se ejecutan a petición de métodos DatabaseMetaData permanecen abiertos después de una operación de confirmación. El tipo de datos de esta propiedad es int.

Cuando una aplicación ejecuta métodos DatabaseMetaData, IBM Data Server Driver para JDBC y SQLJ ejecuta consultas para el catálogo de la fuente de datos de destino. Por omisión, la capacidad de retención de esos cursores es la misma que la capacidad de retención de los cursores de aplicación. Para

utilizar una capacidad de retención diferente para las consultas de catálogo, utilice la propiedad resultSetHoldabilityForCatalogQueries. Los valores posibles son:

DB2BaseDataSource.HOLD_CURSORS_OVER_COMMIT (1)
Dejar abiertos los cursores de las consultas de catálogo después de una operación de confirmación, sin importar el valor de resultSetHoldability.

DB2BaseDataSource.CLOSE_CURSORS_AT_COMMIT (2)
Cerrar los cursores de las consultas de catálogo después de una operación de confirmación, sin importar el valor de resultSetHoldability.

DB2BaseDataSource.NOT_SET (0)
Utilizar el valor de resultSetHoldability para las consultas de catálogo. Éste es el valor por omisión.

returnAlias
Especifica si el controlador JDBC devuelve filas de alias de tabla y sinónimos para métodos DatabaseMetaData que devuelven información de tabla, como por ejemplo getTables. El tipo de datos de returnAlias es int. Los valores posibles son:

0 No devolver filas para alias ni sinónimos de tablas en la salida de métodos DatabaseMetaData que devuelven información de tabla.

1 En el caso de tablas que tienen alias o sinónimos, devolver filas de alias y sinónimos de dichas tablas, así como filas de tablas, en la salida de métodos DatabaseMetaData que devuelven información de tabla. Éste es el valor por omisión.

statementConcentrator
Especifica si IBM Data Server Driver para JDBC y SQLJ emplea la funcionalidad del concentrador de sentencias de la fuente de datos. El concentrador de sentencias es la capacidad de eludir la preparación de una sentencia cuando es igual que una sentencia en la antememoria de sentencias dinámicas, excepto los valores literales. La funcionalidad del concentrador de sentencias se aplica únicamente a las sentencias de SQL que tienen literales pero no tienen marcadores de parámetro. Los valores posibles son:

DB2BaseDataSource.STATEMENT_CONCENTRATOR_OFF (1)
IBM Data Server Driver para JDBC y SQLJ no utiliza la funcionalidad del concentrador de sentencias de la fuente de datos.

DB2BaseDataSource.STATEMENT_CONCENTRATOR_WITH_LITERALS (2)
IBM Data Server Driver para JDBC y SQLJ emplea la funcionalidad de concentrador de sentencias de la fuente de datos.

DB2BaseDataSource.STATEMENT_CONCENTRATOR_NOT_SET (0)
Habilita el comportamiento por omisión del servidor de datos para la funcionalidad del concentrador de sentencias. Éste es el valor por omisión.

En el caso de las fuentes de datos DB2 Database para Linux, UNIX y Windows que dan soporte a la funcionalidad del concentrador de sentencias, la funcionalidad se emplea si el parámetro de configuración STMT_CONC está establecido en ON en la fuente de datos. En caso contrario, no se utiliza la funcionalidad del concentrador de sentencias.

En el caso de las fuentes de datos DB2 para z/OS que dan soporte a la funcionalidad del concentrador de sentencias, la funcionalidad no se utiliza si no se ha determinado statementConcentrator.

streamBufferSize
Especifica el tamaño, en bytes, de los almacenamientos intermedios del controlador JDBC para truncar datos LOB o XML. El controlador JDBC utiliza el valor streamBufferSize independientemente de que utilice la modalidad continua progresiva. El tipo de datos de streamBufferSize es int. El valor por omisión es 1048576.

Si el controlador JDBC utiliza la modalidad continua progresiva, los datos LOB o XML se materializarán si caben en los almacenamientos intermedios y el controlador no utilizará la propiedad fullyMaterializeLobData.

DB2 para z/OS Versión 9.1 y versiones posteriores es compatible con la modalidad continua progresiva para objetos LOB y XML. DB2 Database para Linux, UNIX y Windows Versión 9.5 y posteriores, e IBM Informix Versión 11.50 y posteriores soportan la modalidad continua progresiva para los LOB.

supportsAsynchronousXARollback
Especifica si IBM Data Server Driver para JDBC y SQLJ da soporte a operaciones de retrotracción XA asíncrona. El tipo de datos de esta propiedad es int. El valor por omisión es `DB2BaseDataSource.NO` (2). Si la aplicación se ejecuta en un servidor de aplicaciones BEA WebLogic Server, establezca supportsAsynchronousXARollback en `DB2BaseDataSource.YES` (1).

sysSchema
Especifica el esquema de las tablas o vistas de catálogo de duplicación que se examinan cuando una aplicación invoca un método DatabaseMetaData. La propiedad sysSchema se denominaba cliSchema anteriormente.

timeFormat
Especifica:

- El formato en que se debe especificar el argumento String del método PreparedStatement.setString sobre una columna TIME.
- El formato en que se devuelve el resultado del método ResultSet.getString o CallableStatement.getString sobre una columna TIME.

El tipo de datos de timeFormat es int.

Los valores posibles de timeFormat son:

Constante	Valor entero	Formato
`com.ibm.db2.jcc.DB2BaseDataSource.ISO`	1	*hh:mm:ss*
`com.ibm.db2.jcc.DB2BaseDataSource.USA`	2	*hh:mm* am o *hh:mm* pm
`com.ibm.db2.jcc.DB2BaseDataSource.EUR`	3	*hh.mm.ss*
`com.ibm.db2.jcc.DB2BaseDataSource.JIS`	4	*hh:mm:ss*

El valor por omisión es `com.ibm.db2.jcc.DB2BaseDataSource.ISO`.

timestampOutputType
Especifica si IBM Data Server Driver para JDBC y SQLJ devuelve un objeto java.sql.Timestamp o com.ibm.db2.jcc.DBTimestamp cuando se llama a las interfaces JDBC estándar ResultSet.getTimestamp, CallableStatement.getTimestamp, ResultSet.getObject o CallableStatement.getObject para devolver información de indicación de fecha y hora.

Los valores posibles son:

DB2BaseDataSource.JDBC_TIMESTAMP (1)
IBM Data Server Driver para JDBC y SQLJ devuelve objetos java.sql.Timestamp desde las llamadas a ResultSet.getTimestamp, CallableStatement.getTimestamp, ResultSet.getObject o CallableStatement.getObject.

DB2BaseDataSource.JCC_DBTIMESTAMP (2)
IBM Data Server Driver para JDBC y SQLJ devuelve objetos com.ibm.db2.jcc.DBTimestamp desde las llamadas a ResultSet.getTimestamp, CallableStatement.getTimestamp, ResultSet.getObject o CallableStatement.getObject.

DB2BaseDataSource.NOT_SET (0)
Éste es el comportamiento por omisión.

El comportamiento es el mismo que el de DB2BaseDataSource.JDBC_TIMESTAMP.

useCachedCursor
Especifica si el cursor asociado a objetos PreparedStatement se coloca en la antememoria y se reutiliza en ejecuciones subsiguientes de PreparedStatement. El tipo de datos de useCachedCursor es boolean.

Si useCachedCursor está establecido en `true`, el cursor asociado a objetos PreparedStatement se coloca en la antememoria, lo cual puede mejorar el rendimiento. `true` es el valor por omisión.

Establezca useCachedCursor en `false` si los objetos PreparedStatement acceden a tablas cuyos tipos o longitudes de columna cambian entre una ejecución y otra de esos objetos PreparedStatement.

useIdentityValLocalForAutoGeneratedKeys
Especifica si IBM Data Server Driver para JDBC y SQLJ utiliza únicamente la función incorporada de SQL IDENTITY_VAL_LOCAL para determinar los valores de clave generados automáticamente. El tipo de datos de esta propiedad es booleano. Los valores posibles son:

true Especifica que IBM Data Server Driver para JDBC y SQLJ siempre utilice la función incorporada de SQL IDENTITY_VAL_LOCAL para determinar los valores de clave generados automáticamente. El controlador utiliza IDENTITY_VAL_LOCAL aunque se pueda usar SELECT FROM INSERT.

Especifique `true` si el servidor de datos de destino soporta SELECT FROM INSERT, pero los objetos de destino no lo hacen. Por ejemplo, SELECT FROM INSERT no es válido para una tabla en la que hay un activador definido.

false Especifica que IBM Data Server Driver para JDBC y SQLJ establezca si se utilizará SELECT FROM INSERT o IDENTITY_VAL_LOCAL para determinar claves generadas automáticamente. `false` es el valor por omisión.

useJDBC4ColumnNameAndLabelSemantics
Especifica cómo IBM Data Server Driver para JDBC y SQLJ maneja las etiquetas de columnas en las llamadas a los métodos ResultSetMetaData.getColumnName, ResultSetMetaData.getColumnLabel y ResultSet.findColumn.

Los valores posibles son:

DB2BaseDataSource.YES (1)

IBM Data Server Driver para JDBC y SQLJ utiliza las reglas siguientes, que se ajustan a la especificación JDBC 4.0, para determinar el valor devuelto por ResultSetMetaData.getColumnName, ResultSetMetaData.getColumnLabel y ResultSet.findColumn:

- El nombre de columna devuelto por ResultSetMetaData.getColumnName es el nombre que tiene la columna en la base de datos.
- El nombre de columna devuelto por ResultSetMetaData.getColumnLabel es la etiqueta que está especificada por la cláusula AS de SQL. Si la cláusula AS de SQL no está especificada, la etiqueta es el nombre de la columna.
- ResultSet.findColumn utiliza como entrada la etiqueta de la columna, tal como está especificada por la cláusula AS de SQL. Si la cláusula AS de SQL no está especificada, la etiqueta es el nombre de la columna.
- IBM Data Server Driver para JDBC y SQLJ no utiliza una etiqueta de columna que está asignada por la sentencia LABEL ON de SQL.

Estas reglas son aplicables a IBM Data Server Driver para JDBC y SQLJ versión 3.50 y posterior, para conexiones con los sistemas de bases de datos siguientes:

- DB2 para z/OS Versión 8 o posterior
- DB2 Database para Linux, UNIX y Windows Versión 8.1 o posterior
- DB2 UDB para iSeries V5R3 o posterior

Para las versiones anteriores del controlador o de los sistemas de bases de datos, se aplican las normas para un valor DB2BaseDataSource.NO establecido en useJDBC4ColumnNameAndLabelSemantics, aunque useJDBC4ColumnNameAndLabelSemantics se haya establecido en DB2BaseDataSource.YES.

DB2BaseDataSource.NO (2)

IBM Data Server Driver para JDBC y SQLJ utiliza las reglas siguientes para determinar los valores devueltos por ResultSetMetaData.getColumnName, ResultSetMetaData.getColumnLabel y ResultSet.findColumn:

Si la fuente de datos no es compatible con la sentencia LABEL ON, o la columna de origen no está definida con la sentencia LABEL ON:

- El valor devuelto por ResultSetMetaData.getColumnName es el nombre de la columna obtenido de la base de datos, si no está especificada ninguna cláusula AS de SQL. Si está especificada la cláusula AS de SQL, el valor devuelto es la etiqueta de la columna.
- El valor devuelto por ResultSetMetaData.getColumnLabel es la etiqueta que está especificada por la cláusula AS de SQL. Si la cláusula AS de SQL no está especificada, el valor devuelto es el nombre de la columna.
- ResultSet.findColumn utiliza el nombre de la columna como parámetro de entrada.

Si la columna de origen está definida con la sentencia LABEL ON:

- El valor devuelto por ResultSetMetaData.getColumnName es el nombre de la columna obtenido de la base de datos, si no está especificada ninguna cláusula AS de SQL. Si la cláusula AS de SQL

está especificada, el valor devuelto es la etiqueta de columna que está especificada en la cláusula AS.

- El valor devuelto por ResultSetMetaData.getColumnLabel es la etiqueta que está especificada en la sentencia LABEL ON.
- ResultSet.findColumn utiliza el nombre de la columna como parámetro de entrada.

Estas reglas se ajustan al comportamiento de la versión de IBM Data Server Driver para JDBC y SQLJ anterior a la Versión 3.50.

DB2BaseDataSource.NOT_SET (0)
Éste es el comportamiento por omisión.

Para IBM Data Server Driver para JDBC y SQLJ versión 3.50 y anterior, el comportamiento por omisión para useJDBC4ColumnNameAndLabelSemantics es el mismo que el comportamiento para DB2BaseDataSource.NO.

Para IBM Data Server Driver para JDBC y SQLJ versión 4.0 y posterior:

- El comportamiento por omisión para useJDBC4ColumnNameAndLabelSemantics es el mismo que para DB2BaseDataSource.YES para las conexiones con los siguientes sistemas de bases de datos:
  - DB2 para z/OS Versión 8 o posterior
  - DB2 Database para Linux, UNIX y Windows Versión 8.1 o posterior
  - DB2 UDB para iSeries V5R3 o posterior
- Para las conexiones con versiones anteriores de estos sistemas de bases de datos, el comportamiento por omisión para useJDBC4ColumnNameAndLabelSemantics es DB2BaseDataSource.NO.

formatoXml
Especifica el formato que se utiliza para recuperar datos XML del servidor de datos. El formato XML no se puede modificar después de haber establecido una conexión. Los valores posibles son:

com.ibm.db2.jcc.DB2BaseDataSource.XML_FORMAT_NOT_SET (-Integer.MAX_VALUE)
Especifica que se utiliza el formato XML por omisión. El valor por omisión es el formato XML de texto.

com.ibm.db2.jcc.DB2BaseDataSource.XML_FORMAT_TEXTUAL (0)
Especifica que se utiliza el formato de texto XML.

com.ibm.db2.jcc.DB2BaseDataSource.XML_FORMAT_BINARY (1)
Especifica que se utiliza el formato XML binario.

Cuando se utiliza el formato binario XML, los datos XML que se pasan a IBM Data Server Driver para JDBC y SQLJ no pueden hacer referencia a entidades externas, internas o DTD internos. Sólo se admiten los DTD externos si dichos se han registrado anteriormente en la fuente de datos.

com.ibm.db2.jcc.DB2ConnectionPoolDataSource.maxStatements
Controla una antememoria interna de sentencias que está asociada a un PooledConnection. El tipo de datos de esta propiedad es int. Los valores posibles son:

entero positivo
Habilita la antememoria interna de sentencias para un PooledConnection y especifica el número de sentencias que IBM Data Server Driver para JDBC y SQLJ mantiene abiertas en la antememoria.

0 o entero negativo
Inhabilita el almacenamiento en la antememoria interna de sentencias para el objeto PooledConnection. 0 es el valor por omisión.

maxStatements controla la antememoria interna de sentencias que se asocia con un PooledConnection solamente cuando se crea el objeto PooledConnection. maxStatements no tiene ningún efecto sobre el almacenamiento en antememoria para un objeto PooledConnection ya existente.

maxStatements sólo se aplica a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS y a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4.

Propiedades comunes de IBM Data Server Driver para JDBC y SQLJ para DB2 para z/OS e IBM Informix

Algunas de las propiedades de IBM Data Server Driver para JDBC y SQLJ se aplican a los servidores de datos IBM Informix y DB2 para z/OS.

Las propiedades que se aplican a IBM Informix y DB2 para z/OS son:

enableConnectionConcentrator
Indica si la función del concentrador de conexiones de IBM Data Server Driver para JDBC y SQLJ está habilitada.

El tipo de datos de enableConnectionConcentrator es boolean. El valor por omisión es `false`.

enablConnectionConcentrator es aplicable solamente a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4.

keepDynamic
Especifica si la fuente de datos mantiene sentencias de SQL dinámico ya preparadas en la antememoria de sentencias dinámicas después de puntos de confirmación para que esas sentencias preparadas se puedan reutilizar. El tipo de datos de esta propiedad es int. Los valores válidos son `DB2BaseDataSource.YES` (1) y `DB2BaseDataSource.NO` (2).

Si la propiedad keepDynamic no está especificada, el valor de keepDynamic es `DB2BaseDataSource.NOT_SET` (0). Si la conexión está establecida con un servidor DB2 para z/OS, no se almacenan sentencias dinámicas en la antememoria para una conexión si la propiedad no está establecida. Si la conexión está establecida con una fuente de datos IBM Informix, se almacenan sentencias dinámicas en la antememoria para una conexión si la propiedad no está establecida.

keepDynamic se utiliza con la opción -keepdynamic de DB2Binder. El valor especificado para la propiedad keepDynamic debe coincidir con el valor de -keepdynamic que se especificó al ejecutar DB2Binder.

Para un servidor de bases de datos DB2 para z/OS, solamente se pueden almacenar sentencias dinámicas en la antememoria si la antememoria de sentencias dinámicas de EDM está habilitada en la fuente de datos. El parámetro de subsistema CACHEDYN debe establecerse en `DB2BaseDataSource.YES` para habilitar la antememoria de sentencias dinámicas.

maxTransportObjects
Especifica el número máximo de objetos de transporte que se pueden utilizar para todas las conexiones con el objeto DataSource asociado. IBM Data Server Driver para JDBC y SQLJ utiliza objetos de transporte y una agrupación de objetos de transporte global para dar soporte al concentrador de conexiones y al equilibrado de carga de trabajo Sysplex. Existe un solo objeto de transporte para cada conexión física con la fuente de base de datos.

El tipo de datos de esta propiedad es int.

El valor maxTransportObjects se pasa por alto si las propiedades enableConnectionConcentrator y enableSysplexWLB no se establecen para habilitar el uso del concentrador de conexiones o del equilibrado de carga de trabajo Sysplex.

Si no se ha alcanzado el valor maxTransportObjects y no hay ningún objeto de transporte disponible en la agrupación de objetos de transporte global, la agrupación creará un objeto de transporte nuevo. Si se ha alcanzado el valor maxTransportObjects, la aplicación esperará el tiempo especificado por la propiedad de configuración db2.jcc.maxTransportObjectWaitTime. Una vez transcurrido dicho período de tiempo, si todavía no hay ningún objeto de transporte disponible en la agrupación, la agrupación emitirá una excepción SQLException.

maxTransportObjects **no** altera temporalmente la propiedad de configuración db2.jcc.maxTransportObjects. maxTransportObjects no tiene efecto alguna en las conexiones procedentes de otros objetos DataSource. Si el valor maxTransportObjects es mayor que el valor db2.jcc.maxTransportObjects, maxTransportObjects no incrementará el valor db2.jcc.maxTransportObjects.

Para la Versión 3.63, 4.13 o posterior de IBM Data Server Driver para JDBC y SQLJ, el valor por omisión para maxTransportObjects es 1000. Para las versiones anteriores de IBM Data Server Driver para JDBC y SQLJ, el valor por omisión de maxTransportObjects es -1, lo que significa que el número de objetos de transporte de DataSource está limitado únicamente por el valor db2.jcc.maxTransportObjects del controlador.

Propiedades comunes de IBM Data Server Driver para JDBC y SQLJ para IBM Informix y DB2 Database para Linux, UNIX y Windows

Algunas de las propiedades de IBM Data Server Driver para JDBC y SQLJ se aplican a los servidores de datos IBM Informix y DB2 Database para Linux, UNIX y Windows.

Las propiedades que se aplican a IBM Informix y DB2 Database para Linux, UNIX y Windows son:

currentLockTimeout
Especifica si los servidores DB2 Database para Linux, UNIX y Windows esperan un bloqueo cuando éste no se puede obtener de forma inmediata. El tipo de datos de esta propiedad es int. Los valores posibles son:

entero La espera para un entero *segundos*. *entero* oscila entre -1 y 32767, ambos inclusive.

LOCK_TIMEOUT_NO_WAIT
No espere ningún bloqueo. Éste es el valor por omisión.

LOCK_TIMEOUT_WAIT_INDEFINITELY
Espere un bloqueo durante un tiempo no definido.

LOCK_TIMEOUT_NOT_SET
Especifica utilizar el valor por omisión para la fuente de datos.

Propiedades de IBM Data Server Driver para JDBC y SQLJ para DB2 Database para Linux, UNIX y Windows

Algunas de las propiedades de IBM Data Server Driver para JDBC y SQLJ son aplicables solamente a servidores DB2 Database para Linux, UNIX y Windows.

Estas propiedades son las siguientes:

connectNode
Especifica el servidor de particiones de base de datos de destino al que se conecta una aplicación. El tipo de datos de esta propiedad es int. Este valor puede estar comprendido entre 0 y 999. El valor por omisión es el servidor de particiones de base de datos que está definido con el puerto 0. La propiedad connectNode es aplicable solamente IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 con servidores DB2 Database para Linux, UNIX y Windows.

currentExplainSnapshot
Especifica el valor del registro especial CURRENT EXPLAIN SNAPSHOT. El registro especial CURRENT EXPLAIN SNAPSHOT habilita e inhabilita el recurso de instantáneas Explain. El tipo de datos de esta propiedad es String. La longitud máxima es de ocho bytes. Esta propiedad solamente es aplicable a conexiones con fuentes de datos que son compatibles con el registro especial CURRENT EXPLAIN SNAPSHOT, tales como DB2 Database para Linux, UNIX y Windows.

currentQueryOptimization
Especifica un valor que controla la clase de optimización de consulta que el gestor de la base de datos lleva a cabo cuando vincula sentencias de SQL dinámico. El tipo de datos de esta propiedad es int. Los valores posibles de currentQueryOptimization son:

0 Especifica que se lleva a cabo una optimización mínima para generar un plan de acceso. Esta clase es la más adecuada para el acceso SQL dinámico simple para tablas bien indexadas.

1 Especifica que se realiza una optimización prácticamente igual a la de DB2 Database para Linux, UNIX y Windows Versión 1 para crear un plan de acceso.

2 Especifica un nivel de optimización más alto que el de DB2 Database para Linux, UNIX y Windows Versión 1, pero con un coste de optimización significativamente menor que los niveles 3 y superiores, especialmente para consultas muy complejas.

3 Especifica que se efectúa una optimización moderada para generar un plan de acceso.

5 Especifica que se efectúa una optimización significativa para generar un plan de acceso. Para consultas SQL dinámico complejas, se utilizan las normas heurísticas para limitar el tiempo empleado en seleccionar un plan de acceso. Cuando sea posible, las consultas utilizarán tablas de consulta materializada en lugar de las tablas base subyacentes.

7 Especifica que se efectúa una optimización significativa para generar un plan de acceso. Este valor es igual a 5, pero sin las reglas heurísticas.

9 Especifica la cantidad máxima de optimización que se ejecuta para crear un plan de acceso. Puede expandir mucho el número de planes de acceso posibles que se evalúan. Esta clase debe utilizarse para determinar si se puede generar un plan de acceso mejor para las consultas muy complejas y de muy larga ejecución que utilizan tablas grandes. Las medidas de explicación y de rendimiento se pueden utilizar para verificar que se haya generado el plan mejor.

optimizationProfile
Especifica un perfil de optimización que se utiliza durante la optimización de SQL. El tipo de datos de esta propiedad es String. El valor de optimizationProfile se utiliza para definir el registro especial OPTIMIZATION PROFILE. El valor por omisión es nulo.

La propiedad optimizationProfile es aplicable solamente a servidores DB2 Database para Linux, UNIX y Windows.

optimizationProfileToFlush
Especifica el nombre de un perfil de optimización que se debe eliminar de la antememoria de perfiles de optimización. El tipo de datos de esta propiedad es String. El valor por omisión es nulo.

plugin
Nombre del plugin de seguridad JDBC de la parte del cliente. Esta propiedad es de tipo `Object` y contiene una instancia nueva del método de plugin de seguridad JDBC.

pluginName
Nombre del módulo del plugin de seguridad del lado del servidor.

retryWithAlternativeSecurityMechanism
Especifica si IBM Data Server Driver para JDBC y SQLJ reintenta una conexión con un mecanismo de seguridad alternativo si el mecanismo de seguridad especificado por la propiedad securityMechanism no es compatible con el origen de datos. El tipo de datos de esta propiedad es int. Los valores posibles son:

com.ibm.db2.jcc.DB2BaseDataSource.YES (1)
Reintente la conexión utilizando un mecanismo de seguridad alternativo. IBM Data Server Driver para JDBC y SQLJ emite el código de aviso +4222 y reintenta la conexión con el mecanismo de mayor seguridad disponible.

com.ibm.db2.jcc.DB2BaseDataSource.NO (2) o com.ibm.db2.jcc.DB2BaseDataSource.NOT_SET (0)
No reintente la conexión utilizando un mecanismo de seguridad alternativo.

retryWithAlternativeSecurityMechanism se aplica solamente a las conexiones de IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 con DB2 Database para Linux, UNIX y Windows.

useTransactionRedirect
Especifica si el sistema DB2 encamina sentencias de SQL hacia particiones de base de datos diferentes para lograr un mejor rendimiento. El tipo de datos de esta propiedad es booleano. El valor por omisión es `false`.

Esta propiedad es aplicable solamente cuando se cumplen estas condiciones:

- La conexión se establece con un servidor DB2 Database para Linux, UNIX y Windows que utiliza un entorno de base de datos particionada.
- La clave de particionamiento permanece constante durante toda una transacción.

Si useTransactionRedirect es `true`, IBM Data Server Driver para JDBC y SQLJ envía peticiones de conexión al nodo DPF donde residen los datos de destino de la primera sentencia direccionable de la transacción. Entonces DB2 Database para Linux, UNIX y Windows encamina la sentencia de SQL hacia particiones diferentes según sea necesario.

Propiedades de IBM Data Server Driver para JDBC y SQLJ correspondientes a DB2 para z/OS

Algunas de las propiedades de IBM Data Server Driver para JDBC y SQLJ son aplicables solamente a servidores DB2 para z/OS.

Estas propiedades son las siguientes:

`accountingInterval`
Especifica si se crean registros contables de DB2 en los puntos de confirmación o al concluir la conexión física con la fuente de datos. El tipo de datos de esta propiedad es String.

Si el valor de accountingInterval es "COMMIT" y no hay cursores retenidos que estén abiertos, DB2 escribe un registro contable cada vez que la aplicación confirma trabajo. Si el valor de accountingInterval es "COMMIT" y la aplicación realiza una operación de confirmación mientras hay abierto un cursor retenido, el intervalo contable abarca dicho punto de confirmación y finaliza en el siguiente punto de finalización de intervalo contable válido. Si el valor de accountingInterval no es "COMMIT", los registros contables se crean al concluir la conexión física con la fuente de datos.

La propiedad accountingInterval establece el parámetro *intervalo-contable* para una llamada de inicio de sesión RRSAF subyacente. Si el valor del parámetro de subsistema ACCUMACC no es NO, el valor ACCUMACC altera temporalmente la configuración de accountingInterval.

accountingInterval sólo es aplicable a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS. accountingInterval no es aplicable a las conexiones en CICS o IMS, ni a los procedimientos almacenados Java.

La propiedad accountingInterval altera temporalmente la propiedad de configuración db2.jcc.accountingInterval.

`charOutputSize`
Especifica el número máximo de bytes que se deben utilizar para parámetros de procedimiento almacenado INOUT o OUT que estén registrados como `Types.CHAR`. charOutputSize sólo se aplica a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 con servidores de bases de datos DB2 para z/OS.

Puesto que la información de DESCRIBE para los parámetros de procedimiento almacenado INOUT y OUT no está disponible en tiempo de ejecución, por omisión, IBM Data Server Driver para JDBC y SQLJ establece 32767 como longitud máxima de cada parámetro INOUT o OUT de carácter. Para los procedimientos almacenados con muchos parámetros `Types.CHAR`, este valor máximo puede tener como resultado la asignación de mucho más almacenamiento del necesario.

Para utilizar el almacenamiento de modo más eficaz, establezca charOutputSize con la longitud máxima prevista para cualquier parámetro INOUT o OUT de `Types.CHAR`.

charOutputSize no tiene ningún efecto sobre los parámetros INOUT o OUT registrados como `Types.VARCHAR` o `Types.LONGVARCHAR`. El controlador utiliza la longitud por omisión, 32767, para los parámetros `Types.VARCHAR` y `Types.LONGVARCHAR`.

El valor que seleccione para charOutputSize debe tener en cuenta la posibilidad de ampliación durante la conversión de caracteres. Puesto que IBM Data Server Driver para JDBC y SQLJ no tiene información sobre el CCSID correspondiente al servidor que se utiliza para los valores de los parámetros de salida, el controlador solicita los datos de salida del procedimiento almacenado en UTF-8 Unicode. El valor de charOutputSize debe ser el número máximo de bytes que se necesitan después de que el valor del parámetro se convierta a UTF-8 Unicode. Los caracteres UTF-8 Unicode pueden necesitar hasta tres bytes. (El símbolo del euro es un ejemplo de carácter UTF-8 de tres bytes). Para garantizar que el valor de charOutputSize sea suficientemente elevado, si no dispone de información sobre los datos de salida, establezca como valor de charOutputSize un valor del triple de la longitud definida del parámetro CHAR más elevado.

clientUser
Especifica el nombre de usuario del cliente actual de la conexión. Esta información se utiliza con fines de contabilidad de clientes. A diferencia del nombre de usuario de una conexión JDBC, este valor puede cambiar durante una conexión. Para un servidor DB2 para z/OS, la longitud máxima es 16 bytes.

Esta propiedad solamente es aplicable a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 sobre DB2 para z/OS.

clientWorkstation
Especifica el nombre de la estación de trabajo correspondiente al cliente actual de la conexión. Esta información se utiliza con fines de contabilidad de clientes. Este valor puede cambiar durante una conexión. El tipo de datos de esta propiedad es String. Para un servidor DB2 para z/OS, la longitud máxima es de 18 bytes. Una serie vacía Java ("") es válida para este valor, pero un valor `null` de Java no es válido.

Esta propiedad solamente es aplicable a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 sobre DB2 para z/OS.

currentLocaleLcCtype
Especifica el entorno local LC_CTYPE que se utiliza para ejecutar sentencias de SQL que utilizan una función incorporada que hace referencia a un entorno local. El tipo de datos de esta propiedad es String. Si currentLocaleLcCtype está establecido, IBM Data Server Driver para JDBC y SQLJ establece el registro especial CURRENT LOCALE LC_CTYPE del servidor de datos en el valor de la propiedad. currentLocaleLcCtype no tiene ningún valor por omisión.

currentLocaleLcCtype se puede establecer solamente al inicio de una conexión y no se puede cambiar mientras la conexión esté activa.

currentSQLID
Especifica:

- El ID de autorización que se utiliza para la comprobación de autorizaciones en las sentencias CREATE, GRANT y REVOKE de SQL preparadas dinámicamente.

- El propietario de un espacio de tablas, base de datos, grupo de almacenamiento o sinónimo que es creado por una sentencia CREATE emitida dinámicamente.
- El calificador implícito de todos los nombres de tabla, vista, alias e índice especificados en sentencias de SQL dinámico.

La propiedad currentSQLID define el valor del registro especial CURRENT SQLID en un servidor DB2 para z/OS. Si la propiedad currentSQLID no está definida, el nombre de esquema por omisión es el valor del registro especial CURRENT SQLID.

enableMultiRowInsertSupport
Especifica si IBM Data Server Driver para JDBC y SQLJ utiliza INSERT de varias filas para operaciones INSERT o MERGE por lotes, cuando el servidor de datos de destino es un servidor DB2 para z/OS que soporta las operaciones INSERT de varias filas. Las operaciones por lotes deben ser llamadas PreparedStatement con marcadores de parámetro. El tipo de datos de esta propiedad es booleano. El valor por omisión es `true`.

No se puede cambiar el valor de enableMultiRowInsertSupport mientras dure una conexión. enableMultiRowInsertSupport debe estar establecido en false si las sentencias INSERT FROM SELECT se ejecutan en un lote. En caso contrario, el controlador emite una BatchUpdateException.

jdbcCollection
Especifica el ID de colección de los paquetes que son utilizados por una instancia de IBM Data Server Driver para JDBC y SQLJ durante la ejecución. El tipo de datos de jdbcCollection es String. El valor por omisión es NULLID.

Esta propiedad se utiliza con la opción -collection de DB2Binder. El programa de utilidad DB2Binder debe previamente haber vinculado paquetes de IBM Data Server Driver para JDBC y SQLJ en el servidor utilizando un valor de -collection que coincida con el valor de jdbcCollection.

El valor de jdbcCollection no determina la colección que se utiliza para aplicaciones SQLJ. Para SQLJ, la colección está determinada por la opción -collection del personalizador de SQLJ.

jdbcCollection no es aplicable a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS.

maxConnCachedParamBufferSize
Especifica el tamaño máximo del almacenamiento intermedio interno que se utiliza para almacenar en la antememoria los valores de parámetros de entrada de los objetos PreparedStatement. El almacenamiento intermedio almacena valores en el lado del código nativo procedentes del lado del código Java del controlador de IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS. El almacenamiento intermedio es utilizado por todos los objetos PreparedStatement para Connection. El valor por omisión es 1048576 (1 MB). El valor por omisión debe ser adecuado para la mayoría de los usuarios. Establezca maxConnCachedParamBufferSize en un valor mayor si muchas de las aplicaciones que se ejecutan bajo la instancia del controlador tienen objetos PreparedStatement con grandes cantidades de parámetros de entrada o parámetros de entrada grandes. El valor maxConnCachedParamBufferSize debe ser mayor que el tamaño máximo de todos los datos de parámetros de entrada para Connection. Sin embargo, también debe tener en cuenta el número total de conexiones y la cantidad máxima de memoria que está disponible cuando se establece el valor maxConnCachedParamBufferSize.

El almacenamiento intermedio existe mientras dure un objeto Connection, a menos que alcance el tamaño máximo. Si esto sucede, el almacenamiento intermedio se libera en cada llamada al código nativo. El almacenamiento intermedio correspondiente en el lado del código Java se libera en las llamadas PreparedStatement.clearParameters y PreparedStatement.close. Los almacenamientos intermedios no se borran si una aplicación llama a PreparedStatement.clearParameters y los almacenamientos intermedios no han alcanzado el tamaño máximo.

maxRowsetSize
Especifica el número máximo de bytes que se utilizan para la puesta en almacenamiento intermedio de rowset para cada sentencia, cuando IBM Data Server Driver para JDBC y SQLJ utiliza operaciones FETCH de varias filas para cursores. El tipo de datos de esta propiedad es int. El valor por omisión es 32767.

maxRowsetSize es aplicable solamente a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS.

reportLongTypes
Especifica si los métodos DatabaseMetaData informan de tipos de datos de columna LONG VARCHAR y LONG VARGRAPHIC como tipos de datos long. El tipo de datos de esta propiedad es short. Los valores posibles son:

com.ibm.db2.jcc.DB2BaseDataSource.NO (2) o com.ibm.db2.jcc.DB2BaseDataSource.NOT_SET (0)
Especifica que los métodos DatabaseMetaData que devuelven información sobre una columna LONG VARCHAR o LONG VARGRAPHIC devuelven java.sql.Types.VARCHAR en la columna DATA_TYPE y VARCHAR o VARGRAPHIC en la columna TYPE_NAME del conjunto de resultados. Éste es el valor por omisión para DB2 para z/OS versión 9 o posterior.

com.ibm.db2.jcc.DB2BaseDataSource.YES (1)
Especifica que los métodos DatabaseMetaData que devuelven información sobre una columna LONG VARCHAR o LONG VARGRAPHIC devuelven java.sql.Types.LONGVARCHAR en la columna DATA_TYPE y LONG VARCHAR o LONG VARGRAPHIC en la columna TYPE_NAME del conjunto de resultados.

sendCharInputsUTF8
Especifica si IBM Data Server Driver para JDBC y SQLJ convierte los datos de entrada de tipo carácter al CCSID del servidor de bases de datos DB2 para z/OS, o si envía los datos en la codificación UTF-8 para su conversión por el servidor de bases de datos. La propiedad sendCharInputsUTF8 es aplicable solamente a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 con servidores de bases de datos DB2 para z/OS. El tipo de datos de esta propiedad es int. Si esta propiedad está definida también a nivel de controlador (db2.jcc.sendCharInputsUTF8), este valor prevalece sobre el valor a nivel de controlador.

Los valores posibles son:

com.ibm.db2.jcc.DB2BaseDataSource.NO (2)
Especifica que IBM Data Server Driver para JDBC y SQLJ convierte los datos de entrada de tipo carácter a la codificación de destino antes de enviar los datos al servidor de bases de datos DB2 para z/OS. com.ibm.db2.jcc.DB2BaseDataSource.NO es el valor por omisión.

com.ibm.db2.jcc.DB2BaseDataSource.YES (1)
Especifica que IBM Data Server Driver para JDBC y SQLJ envía los datos de entrada de tipo carácter al servidor de bases de datos DB2 para z/OS en la codificación UTF-8. El servidor de bases de datos convierte los datos desde la codificación UTF-8 al CCSID de destino.

Especifique com.ibm.db2.jcc.DB2BaseDataSource.YES solamente si la conversión al CCSID de destino por el SDK de Java causa problemas de conversión de caracteres. El problema más habitual se produce cuando se utiliza IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 para insertar un carácter Unicode de salto de línea (U+000A) en una columna de tabla cuyo CCSID es 37, y luego se recuperan esos datos a partir de un cliente que no es un cliente z/OS. Si el SDK de Java realiza la conversión durante la inserción del carácter en la columna, el carácter de salto de línea se convierte en el carácter de línea nueva de EBCDIC, X'15'. Pero, durante la recuperación de los datos, algunos SDK de Java que se ejecutan en sistemas operativos distintos de z/OS convierten el carácter X'15' al carácter de línea siguiente de Unicode (U+0085), en lugar de convertirlo al carácter de salto de línea (U+000A). El carácter de línea siguiente produce un comportamiento inesperado en algunos analizadores sintácticos de XML. Si establece sendCharInputsUTF8 en com.ibm.db2.jcc.DB2BaseDataSource.YES, el servidor de bases de datos DB2 para z/OS convierte el carácter U+000A al carácter de salto de línea de EBCDIC (X'25') durante la inserción en la columna, por lo que el carácter se recupera siempre como carácter de salto de línea.

La conversión de datos al CCSID de destino en el servidor de bases de datos puede hacer que IBM Data Server Driver para JDBC y SQLJ utilice más memoria que la conversión por el controlador. El controlador asigna memoria para la conversión de datos de tipo carácter desde la codificación de origen a la codificación de los datos que envía al servidor de bases de datos. La cantidad de espacio que el controlador asigna para datos de tipo carácter que se envían a una columna de tabla está basada en la longitud máxima posible de los datos. Los datos con codificación UTF-8 pueden necesitar hasta tres bytes para cada carácter. Por tanto, si el controlador envía datos UTF-8 al servidor de bases de datos, el controlador necesita asignar un espacio tres veces mayor que el número máximo de caracteres de los datos de entrada. Si el controlador realiza la conversión y el CCSID de destino es un CCSID de un solo byte, el controlador necesita asignar solamente un espacio igual al número máximo de caracteres de los datos de entrada.

sessionTimeZone
Especifica el valor para el registro especial CURRENT SESSION TIME ZONE. El tipo de datos de esta propiedad es String.

El valor sessionTimeZone es un valor de huso horario con el formato *sth:tm*. *s* es la señal, *th* es la hora y *tm* los minutos del huso horario. El rango de valores válidos es de -12:59 a +14:00.

sqljEnableClassLoaderSpecificProfiles
Especifica si IBM Data Server Driver para JDBC y SQLJ permite utilizar y cargar perfiles SQLJ con el mismo nombre Java en varios archivos de aplicación J2EE (.ear). El tipo de datos de esta propiedad es booleano. El valor

por omisión es `false`. sqljEnableClassLoaderSpecificProfiles es una propiedad de DataSource. Esta propiedad está pensada principalmente para utilizarla con WebSphere Application Server.

ssid
Especifica el nombre del subsistema DB2 para z/OS local con el que se establece una conexión utilizando IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS. El tipo de datos de esta propiedad es String.

La propiedad ssid altera temporalmente la propiedad de configuración db2.jcc.ssid.

ssid puede ser el nombre de un subsistema local o un nombre de enlace de grupos o de subgrupos.

La especificación de un nombre de subsistema local permite acceder a más de un subsistema en un único LPAR como subsistema local para conexiones que utilizan IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2.

La especificación de un nombre de enlace de grupos o de subgrupos permite el proceso de migración tras error si falla un miembro del grupo de compartimiento de datos. Si falla el subsistema DB2 que está conectado a una aplicación, finaliza la conexión. No obstante, cuando nuevas conexiones utilizan dicho nombre de enlace de grupos o de subgrupos, DB2 para z/OS utiliza un proceso de enlace de grupos o de subgrupos para buscar un subsistema DB2 activo al que conectarse.

ssid sólo se aplica a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 con DB2 para z/OS.

useRowsetCursor
Especifica si IBM Data Server Driver para JDBC y SQLJ utiliza siempre la operación FETCH de varia filas para los cursores desplazables si la fuente de datos soporta la operación FETCH de varia filas. El tipo de datos de esta propiedad es booleano.

Esta propiedad es aplicable solamente a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4, o a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 con DB2 para z/OS. Si la propiedad enableRowsetSupport no está definida, el valor por omisión de useRowsetCursor es `true`. Si la propiedad enableRowsetSupport está definida, la propiedad useRowsetCursor no se utiliza.

Las aplicaciones que hacen uso de la técnica de JDBC 1 para ejecutar operaciones de actualización o supresión de posición deben establecer useRowSetCursor en `false`. Esas aplicaciones no operan debidamente si IBM Data Server Driver para JDBC y SQLJ utiliza sentencias FETCH de varias filas.

Propiedades de IBM Data Server Driver para JDBC y SQLJ correspondientes a IBM Informix

Algunas de las propiedades de IBM Data Server Driver para JDBC y SQLJ son aplicables solamente a bases de datos IBM Informix. Estas propiedades corresponden a variables de entorno de IBM Informix.

Las propiedades que se muestran en letras mayúsculas en la información siguiente se deben especificar en mayúsculas. Para esas propiedades, los métodos getXXX y setXXX se forman añadiendo como prefijo get o set al nombre de la propiedad escrito en mayúsculas. Por ejemplo:

```
boolean dbDate = DB2BaseDateSource.getDBDATE();
```

Las propiedades específicas de IBM Informix son:

DBANSIWARN
Especifica si IBM Data Server Driver para JDBC y SQLJ indica a la base de datos IBM Informix si debe devolver un SQLWarning a la aplicación si una sentencia de SQL utiliza una sintaxis que no sigue el estándar ANSI. El tipo de datos de esta propiedad es booleano. Los valores posibles son:

false ó 0
No envía un valor a la base de datos IBM Informix que indica a la base de datos si debe devolver un SQLWarning a la aplicación si una sentencia de SQL utiliza una sintaxis que no sigue el estándar ANSI. Éste es el valor por omisión.

true ó 1
Envía un valor a la base de datos IBM Informix que indica a la base de datos IDS que debe devolver un SQLWarning a la aplicación si una sentencia de SQL utiliza una sintaxis que no sigue el estándar ANSI.

Existe la posibilidad de utilizar la propiedad DBANSIWARN de IBM Data Server Driver para JDBC y SQLJ para establecer la propiedad DBANSIWARN de IBM Informix, sin embargo, no se puede utilizar la propiedad DBANSIWARN de IBM Data Server Driver para JDBC y SQLJ para restablecer la propiedad DBANSIWARN de IBM Informix.

DBDATE
Especifica el formato de usuario final para los valores de tipo DATE. El tipo de datos de esta propiedad es String. Los valores posibles están descritos en la explicación de la variable de entorno DBDATE, en el manual *IBM Informix Guide to SQL: Reference*.

El valor por omisión es "Y4MD-".

DBPATH
Especifica una lista de valores, separados por signos de dos puntos, que identifican los servidores de bases de datos que contienen bases de datos. El tipo de datos de esta propiedad es String. Cada valor puede ser:
- Una vía de acceso completa
- Una vía de acceso relativa
- El nombre de un servidor de bases de datos IBM Informix
- Un nombre de servidor y una vía de acceso completa

El valor por omisión es ".".

DBSPACETEMP
Especifica una lista de espacios de base de datos existentes, separados por comas o signos de dos puntos, donde se colocan tablas temporales. El tipo de datos de esta propiedad es String.

Si esta propiedad no está definida, no se envía ningún valor al servidor. Se utiliza el valor de la variable de entorno DBSPACETEMP.

DBTEMP
Especifica la vía de acceso completa de un directorio existente donde se colocan archivos temporales y tablas temporales. El tipo de datos de esta propiedad es String. El valor por omisión es "/tmp".

DBUPSPACE
Especifica la cantidad máxima de espacio de disco del sistema y la cantidad máxima de memoria, en kilobytes, que la sentencia UPDATE STATISTICS

puede utilizar cuando crea varias distribuciones de columnas al mismo tiempo. El tipo de datos de esta propiedad es String.

El formato de DBUPSPACE es "*espacio-disco-máximo*:*memoria-máxima*".

Si esta propiedad no está definida, no se envía ningún valor al servidor. Se utiliza el valor de la variable de entorno DBUPSPACE.

DB_LOCALE
Especifica el entorno local de la base de datos, que el servidor de bases de datos utiliza para procesar datos sensibles al entorno local. El tipo de datos de esta propiedad es String. Los valores válidos son los mismos que los valores válidos para la variable de entorno DB_LOCALE. El valor por omisión es nulo.

DELIMIDENT
Especifica si se pueden utilizar identificadores de SQL delimitados en una aplicación. El tipo de datos de esta propiedad es booleano. Los valores posibles son:

false La aplicación no puede contener identificadores de SQL delimitados. Se utilizan comillas dobles (") o comillas simples (') para delimitar las series literales. Éste es el valor por omisión.

true La aplicación puede contener identificadores de SQL delimitados. Los identificadores de SQL delimitados se deben encerrar entre comillas dobles. Se utilizan comillas simples (') para delimitar series literales.

IFX_DIRECTIVES
Especifica si el optimizador permite directivas de optimización de consultas desde dentro de una consulta. El tipo de datos de esta propiedad es String. Los valores posibles son:

"1" u "ON"
Se aceptan las directivas de optimización.

"0" u "OFF"
No se aceptan las directivas de optimización.

Si esta propiedad no está definida, no se envía ningún valor al servidor. Se utiliza el valor de la variable de entorno IFX_DIRECTIVES.

IFX_EXTDIRECTIVES
Especifica si el optimizador permite que directivas externas de optimización de consultas procedentes de la tabla de catálogo del sistema sysdirectives se apliquen a las consultas en las aplicaciones existentes. Los valores posibles son:

"1" u "ON"
Se aceptan las directivas externas de optimización de consultas.

"0" u "OFF"
No se aceptan las directivas externas de optimización de consultas.

Si esta propiedad no está definida, no se envía ningún valor al servidor. Se utiliza el valor de la variable de entorno IFX_EXTDIRECTIVES.

IFX_UPDDESC
Especifica si está permitida una operación DESCRIBE de sentencia UPDATE. El tipo de datos de esta propiedad es String.

Cualquier valor no nulo indica que una operación DESCRIBE de una sentencia UPDATE está permitida. El valor por omisión es "1".

IFX_XASTDCOMPLIANCE_XAEND
Especifica si las transacciones globales solamente se liberan después de una

retrotracción explícita o después de cualquier retrotracción. El tipo de datos de esta propiedad es String. Los valores posibles son:

"0" Las transacciones globales se liberan solamente después de una retrotracción explícita. Este comportamiento se ajusta al estándar XA de X/Open.

"1" Las transacciones globales se liberan después de cualquier retrotracción.

Si esta propiedad no está definida, no se envía ningún valor al servidor. Se utiliza el valor de la variable de entorno IFX_XASTDCOMPLIANCE_XAEND.

INFORMIXOPCACHE

Especifica el tamaño de la antememoria, en kilobytes, para el espacio de BLOB del área de transferencia de datos de la aplicación cliente. El tipo de datos de esta propiedad es String. El valor "0" significa que la antememoria no se utiliza,

Si esta propiedad no está definida, no se envía ningún valor al servidor. Se utiliza el valor de la variable de entorno INFORMIXOPCACHE.

INFORMIXSTACKSIZE

Especifica el tamaño de pila, en kilobytes, que el servidor de bases de datos utiliza para la hebra primaria de una sesión cliente. El tipo de datos de esta propiedad es String.

Si esta propiedad no está definida, no se envía ningún valor al servidor. Se utiliza el valor de la variable de entorno INFORMIXSTACKSIZE.

NODEFDAC

Especifica si el servidor de bases de datos impide que se otorguen a PUBLIC privilegios por omisión sobre tablas (SELECT, INSERT, UPDATE y DELETE) cuando se crea una nueva tabla durante la sesión actual, en una base de datos que no se ajusta al estándar ANSI. El tipo de datos de esta propiedad es String. Los valores posibles son:

"yes" El servidor de bases de datos impide que se otorguen a PUBLIC privilegios por omisión sobre tablas cuando se crea una nueva tabla durante la sesión actual, en una base de datos que no se ajusta al estándar ANSI.

"no" El servidor de bases de datos no impide que se otorguen a PUBLIC privilegios por omisión sobre tablas cuando se crea una nueva tabla durante la sesión actual, en una base de datos que no se ajusta al estándar ANSI. Éste es el valor por omisión.

OPTCOMPIND

Especifica el método preferido para realizar una operación de unión en un par ordenado de tablas. El tipo de datos de esta propiedad es String. Los valores posibles son:

"0" El optimizador elige realizar una unión de bucle anidado, cuando sea posible, en lugar de una unión de clasificación-fusión o una unión aleatoria.

"1" Cuando el nivel de aislamiento es lectura repetitiva, el optimizador elige realizar una unión de bucle anidado, cuando sea posible, en lugar de una unión de clasificación-fusión o una unión aleatoria. Cuando el nivel de aislamiento no es lectura repetitiva, el optimizador elige un método de unión basándose en los costes.

"2" El optimizador elige un método de unión basándose en los costes, sin importar la modalidad de aislamiento de la transacción.

Si esta propiedad no está definida, no se envía ningún valor al servidor. Se utiliza el valor de la variable de entorno OPTCOMPIND.

OPTOFC

Especifica si se debe habilitar la funcionalidad optimize-OPEN-FETCH-CLOSE. El tipo de datos de esta propiedad es String. Los valores posibles son:

"0" Inhabilitar la funcionalidad optimize-OPEN-FETCH-CLOSE para todas las hebras de aplicaciones.

"1" Habilitar la funcionalidad optimize-OPEN-FETCH-CLOSE para todos los cursores en todas las hebras de aplicaciones.

Si esta propiedad no está definida, no se envía ningún valor al servidor. Se utiliza el valor de la variable de entorno OPTOFCD.

PDQPRIORITY

Especifica el grado de paralelismo utilizado por el servidor de bases de datos. El valor de PDQPRIORITY determina cómo el servidor de bases de datos asigna recursos, tales como memoria, procesadores y lecturas de disco. El tipo de datos de esta propiedad es String. Los valores posibles son:

"HIGH"
Cuando el servidor de bases de datos asigna recursos entre todos los usuarios, proporciona tantos recursos como sea posible a las consultas.

"LOW" o "1"
El servidor de bases de datos recupera valores de tablas fragmentadas en paralelo.

"OFF" o "0"
El proceso en paralelo está inhabilitado.

Si esta propiedad no está definida, no se envía ningún valor al servidor. Se utiliza el valor de la variable de entorno PDQPRIORITY.

PSORT_DBTEMP

Especifica la vía de acceso completa de un directorio en el que el servidor de bases de datos escribe archivos temporales que se utilizan para una operación de clasificación. El tipo de datos de esta propiedad es String.

Si esta propiedad no está definida, no se envía ningún valor al servidor. Se utiliza el valor de la variable de entorno PSORT_DBTEMP.

PSORT_NPROCS

Especifica el número máximo de hebras que el servidor de bases de datos puede utilizar para clasificar una consulta. El tipo de datos de esta propiedad es String. El valor máximo de PSORT_NPROCS es "10".

Si esta propiedad no está definida, no se envía ningún valor al servidor. Se utiliza el valor de la variable de entorno PSORT_NPROCS.

STMT_CACHE

Especifica si se habilita la antememoria de sentencias compartida. El tipo de datos de esta propiedad es String. Los valores posibles son:

"0" Se inhabilita la antememoria de sentencias compartida.

"1" Se habilita una antememoria de sentencias compartida de 512 KB.

Si esta propiedad no está definida, no se envía ningún valor al servidor. Se utiliza el valor de la variable de entorno STMT_CACHE.

dumpPool

Especifica los tipos de estadísticas que se escriben sobre los sucesos de la agrupación de transporte global, además de las estadísticas de resumen. La agrupación de transporte global se utiliza para el concentrador de conexiones y el equilibrado de la carga de trabajo Sysplex.

El tipo de datos de dumpPool es int. Las propiedades dumpPoolStatisticsOnSchedule y dumpPoolStatisticsOnScheduleFile también se deben definir para escribir estadísticas para poder recoger cualquier estadística.

Con la propiedad db2.jcc.dumpPool se pueden especificar uno o más de los tipos de estadísticas siguientes:

- `DUMP_REMOVE_OBJECT` (hexadecimal: X'01', decimal: 1)
- `DUMP_GET_OBJECT` (hexadecimal: X'02', decimal: 2)
- `DUMP_WAIT_OBJECT` (hexadecimal: X'04', decimal: 4)
- `DUMP_SET_AVAILABLE_OBJECT` (hexadecimal: X'08', decimal: 8)
- `DUMP_CREATE_OBJECT` (hexadecimal: X'10', decimal: 16)
- `DUMP_SYSPLEX_MSG` (hexadecimal: X'20', decimal: 32)
- `DUMP_POOL_ERROR` (hexadecimal: X'80', decimal: 128)

Para rastrear más de un tipo de suceso, añada los valores correspondientes a los tipos de sucesos que desee rastrear. Por ejemplo, suponga que desea rastrear los sucesos DUMP_GET_OBJECT y DUMP_CREATE_OBJECT. Los equivalentes numéricos de estos valores son 2 y 16, por lo que debe especificar 18 para el valor de dumpPool.

El valor por omisión es 0, que significa que sólo se graban las estadísticas de resumen correspondientes a la agrupación de transporte global.

Esta propiedad no tiene un método setXXX ni un método getXXX.

dumpPoolStatisticsOnSchedule

Especifica la frecuencia, en segundos, con que las estadísticas de la agrupación de transporte global se escriben en el archivo especificado por dumpPoolStatisticsOnScheduleFile. La agrupación de objetos de transporte global se utiliza para el concentrador de conexiones y el equilibrado de la carga de trabajo Sysplex.

El valor por omisión es -1. -1 significa que no se graban las estadísticas de agrupación de transporte global.

Esta propiedad no tiene un método setXXX ni un método getXXX.

dumpPoolStatisticsOnScheduleFile

Especifica el nombre del archivo en el que se escriben las estadísticas de la agrupación de transporte global. La agrupación de transporte global se utiliza para el concentrador de conexiones y el equilibrado de la carga de trabajo Sysplex.

Si no se especifica dumpPoolStatisticsOnScheduleFile, no se escriben estadísticas de la agrupación de transporte global.

Esta propiedad no tiene un método setXXX ni un método getXXX.

maxTransportObjectIdleTime

Especifica la cantidad de tiempo en segundos que un objeto de transporte no utilizado debe permanecer en una agrupación de objetos de transporte global para poderlo suprimir de la agrupación. Los objetos de transporte se utilizan para el concentrador de conexiones y el equilibrado de la carga de trabajo Sysplex.

El valor por omisión para maxTransportObjectIdleTime es 10. Si maxTransportObjectIdleTime se establece en un valor menor que 0, los objetos de transporte no utilizados se suprimen de la agrupación inmediatamente. Esta acción **no** es recomendable, ya que puede provocar una importante disminución del rendimiento.

Esta propiedad no tiene un método setXXX ni un método getXXX.

maxTransportObjectWaitTime
Especifica la cantidad de tiempo en segundos que una aplicación espera un objeto de transporte si se alcanza el valor de maxTransportObjects. Los objetos de transporte se utilizan para el concentrador de conexiones y el equilibrado de la carga de trabajo Sysplex. Cuando una aplicación espera más tiempo que el especificado por el valor de maxTransportObjectWaitTime, la agrupación de objetos de transporte global emite una excepción de SQL (SQLException).

El valor por omisión para maxTransportObjectWaitTime es 1. Cualquier valor negativo significa que las aplicaciones esperan indefinidamente.

Esta propiedad no tiene un método setXXX ni un método getXXX.

minTransportObjects
Especifica el límite inferior del número de objetos de transporte de una agrupación de objetos de transporte global para el concentrador de conexiones y el equilibrado de carga de trabajo Sysplex. Cuando se crea una JVM, no hay objetos de transporte en la agrupación. Los objetos de transporte se añaden a la agrupación a medida que se necesitan. Cuando se alcanza el valor de minTransportObjects, el número de objetos de transporte contenidos en la agrupación de objetos de transporte global nunca pasa a ser menor que el valor de minTransportObjects durante la vigencia de esa JVM.

El valor por omisión de minTransportObjects es 0. Cualquier valor que sea menor o igual que 0 significa que la agrupación de objetos de transporte global puede llegar a estar vacía.

Esta propiedad no tiene un método setXXX ni un método getXXX.

Propiedades de configuración de IBM Data Server Driver para JDBC y SQLJ

Las propiedades de configuración de IBM Data Server Driver para JDBC y SQLJ tienen un ámbito a nivel de controlador.

La tabla siguiente resume las propiedades de configuración y las correspondientes propiedades de Connection o DataSource, si existen.

Tabla 56. Resumen de las propiedades de configuración y las correspondientes propiedades de Connection y DataSource

Nombre de la propiedad de configuración	Nombre de la propiedad de Connection o DataSource: com.ibm.db2.jcc.DB2BaseDataSource. ...	Notas
db2.jcc.accountingInterval	accountingInterval	1 en la página 409, 4 en la página 409
db2.jcc.allowSqljDuplicateStaticQueries		4 en la página 409
db2.jcc.charOutputSize	charOutputSize	1 en la página 409, 4 en la página 409
db2.jcc.currentSchema	currentSchema	1 en la página 409, 4 en la página 409, 6 en la página 409

Tabla 56. Resumen de las propiedades de configuración y las correspondientes propiedades de Connection y DataSource (continuación)

Nombre de la propiedad de configuración	Nombre de la propiedad de Connection o DataSource: com.ibm.db2.jcc.DB2BaseDataSource. ...	Notas
db2.jcc.override.currentSchema	currentSchema	2 en la página 409, 4 en la página 409, 6 en la página 409
db2.jcc.currentSQLID	currentSQLID	1 en la página 409, 4 en la página 409
db2.jcc.override.currentSQLID	currentSQLID	2 en la página 409, 4 en la página 409
db2.jcc.decimalRoundingMode	decimalRoundingMode	1 en la página 409, 4 en la página 409, 6 en la página 409
db2.jcc.override.decimalRoundingMode	decimalRoundingMode	2 en la página 409, 4 en la página 409, 6 en la página 409
db2.jcc.defaultSQLState		4 en la página 409
db2.jcc.disableSQLJProfileCaching		4 en la página 409
db2.jcc.dumpPool	dumpPool	1 en la página 409, 3 en la página 409, 4 en la página 409, 5 en la página 409
db2.jcc.dumpPoolStatisticsOnSchedule	dumpPoolStatisticsOnSchedule	1 en la página 409, 3 en la página 409, 4 en la página 409, 5 en la página 409
db2.jcc.dumpPoolStatisticsOnScheduleFile	dumpPoolStatisticsOnScheduleFile	1 en la página 409, 3 en la página 409, 4 en la página 409, 5 en la página 409
db2.jcc.enableInetAddressGetHostName		4 en la página 409, 5 en la página 409, 6 en la página 409
db2.jcc.override.enableMultirowInsertSupport	enableMultirowInsertSupport	2 en la página 409, 4 en la página 409
db2.jcc.encryptionAlgorithm	encryptionAlgorithm	1 en la página 409, 4 en la página 409, 6 en la página 409
db2.jcc.override.encryptionAlgorithm	encryptionAlgorithm	2 en la página 409, 4 en la página 409, 6 en la página 409
db2.jcc.jmxEnabled		4 en la página 409, 5 en la página 409, 6 en la página 409
db2.jcc.lobOutputSize		4 en la página 409
db2.jcc.maxConnCachedParamBufferSize	maxConnCachedParamBufferSize	1 en la página 409, 4 en la página 409
db2.jcc.maxRefreshInterval		4 en la página 409, 5 en la página 409, 6 en la página 409
db2.jcc.maxTransportObjectIdleTime		1 en la página 409, 4 en la página 409, 5 en la página 409, 6 en la página 409
db2.jcc.maxTransportObjectWaitTime		1 en la página 409, 4 en la página 409, 5 en la página 409, 6 en la página 409
db2.jcc.maxTransportObjects	maxTransportObjects	1 en la página 409, 4 en la página 409, 5 en la página 409, 6 en la página 409

Tabla 56. Resumen de las propiedades de configuración y las correspondientes propiedades de Connection y DataSource (continuación)

Nombre de la propiedad de configuración	Nombre de la propiedad de Connection o DataSource: com.ibm.db2.jcc.DB2BaseDataSource. ...	Notas
db2.jcc.minTransportObjects		1 en la página 409, 4 en la página 409, 5 en la página 409, 6 en la página 409
db2.jcc.outputDirectory		6 en la página 409
db2.jcc.pkList	pkList	1 en la página 409, 4 en la página 409
db2.jcc.planName	planName	1 en la página 409, 4 en la página 409
db2.jcc.progressiveStreaming	progressiveStreaming	1 en la página 409, 4 en la página 409, 5 en la página 409, 6 en la página 409
db2.jcc.override.progressiveStreaming	progressiveStreaming	2 en la página 409, 4 en la página 409, 5 en la página 409, 6 en la página 409
db2.jcc.rollbackOnShutdown		4 en la página 409
db2.jcc.securityMechanism	securityMechanism	1 en la página 409, 4 en la página 409, 5 en la página 409, 6 en la página 409
db2.jcc.override.securityMechanism	securityMechanism	2 en la página 409, 4 en la página 409, 5 en la página 409, 6 en la página 409
db2.jcc.sendCharInputsUTF8	sendCharInputsUTF8	4 en la página 409
db2.jcc.sqljToolsExitJVMOnCompletion		4 en la página 409, 6 en la página 409
db2.jcc.sqljUncustomizedWarningOrException		4 en la página 409, 6 en la página 409
db2.jcc.ssid	ssid	1 en la página 409, 4 en la página 409
db2.jcc.traceDirectory	traceDirectory	1 en la página 409, 4 en la página 409, 5 en la página 409, 6 en la página 409
db2.jcc.override.traceDirectory	traceDirectory	2 en la página 409, 4 en la página 409, 5 en la página 409, 6 en la página 409
db2.jcc.traceFile	traceFile	1 en la página 409, 4 en la página 409, 5 en la página 409, 6 en la página 409
db2.jcc.override.traceFile	traceFile	2 en la página 409, 4 en la página 409, 5 en la página 409, 6 en la página 409
db2.jcc.traceFileAppend	traceFileAppend	1 en la página 409, 4 en la página 409, 5 en la página 409, 6 en la página 409
db2.jcc.override.traceFileAppend	traceFileAppend	2 en la página 409, 4 en la página 409, 5 en la página 409, 6 en la página 409
db2.jcc.traceFileCount	traceFileCount	1 en la página 409, 4 en la página 409, 5 en la página 409, 6 en la página 409
db2.jcc.traceFileSize	traceFileSize	1 en la página 409, 4 en la página 409, 5 en la página 409, 6 en la página 409

Tabla 56. Resumen de las propiedades de configuración y las correspondientes propiedades de Connection y DataSource (continuación)

Nombre de la propiedad de configuración	Nombre de la propiedad de Connection o DataSource: com.ibm.db2.jcc.DB2BaseDataSource. ...	Notas
db2.jcc.traceLevel	traceLevel	1, 4, 5, 6
db2.jcc.override.traceLevel	traceLevel	2, 4, 5, 6
db2.jcc.traceOption	traceOption	1, 4, 5, 6
db2.jcc.tracePolling		4, 5, 6
db2.jcc.tracePollingInterval		4, 5, 6
db2.jcc.t2zosTraceFile		4
db2.jcc.t2zosTraceBufferSize		4
db2.jcc.t2zosTraceWrap		4
db2.jcc.useCcsid420ShapedConverter		4

Nota:

1. El valor de la propiedad de Connection o DataSource prevalece sobre el valor de la propiedad de configuración. La propiedad de configuración proporciona un valor por omisión para la propiedad de Connection o DataSource.
2. El valor de la propiedad de configuración altera temporalmente el valor de la propiedad de Connection o DataSource.
3. El correspondiente valor de Connection o DataSource se define solamente para IBM Informix.
4. La propiedad de configuración es aplicable a DB2 para z/OS.
5. La propiedad de configuración es aplicable a IBM Informix.
6. La propiedad de configuración es aplicable a DB2 Database para Linux, UNIX y Windows.

El significado de las propiedades de configuración es el siguiente:

db2.jcc.accountingInterval
Especifica si se crean registros contables de DB2 en los puntos de confirmación o al concluir la conexión física con la fuente de datos. Si el valor de db2.jcc.accountingInterval es COMMIT, se crean registros contables de DB2 en los puntos de confirmación. Por ejemplo:

```
db2.jcc.accountingInterval=COMMIT
```

De lo contrario, se generan registros contables al finalizar la conexión física con la fuente de datos.

db2.jcc.accountingInterval solo es aplicable a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS. db2.jcc.accountingInterval no es aplicable a las conexiones en CICS o IMS, ni a los procedimientos almacenados Java.

Puede alterar temporalmente db2.jcc.accountingInterval estableciendo la propiedad `accountingInterval` para un objeto Connection o DataSource.

Esta propiedad de configuración se aplica únicamente a DB2 para z/OS.

db2.jcc.allowSqljDuplicateStaticQueries
Especifica si se permiten varios iteradores abiertos en una misma sentencia SELECT dentro de una aplicación SQLJ cuando se utiliza IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2.

Para habilitar este soporte, establezca db2.jcc.allowSqljDuplicateStaticQueries en YES o true.

db2.jcc.charOutputSize
Especifica el número máximo de bytes que se deben utilizar para parámetros de procedimiento almacenado INOUT u OUT que estén registrados como `Types.CHAR`.

Puesto que la información de DESCRIBE para los parámetros de procedimiento almacenado INOUT y OUT no está disponible en tiempo de ejecución, por omisión, IBM Data Server Driver para JDBC y SQLJ establece 32767 como longitud máxima de cada parámetro INOUT o OUT de carácter. Para los procedimientos almacenados con muchos parámetros `Types.CHAR`, este valor máximo puede tener como resultado la asignación de mucho más almacenamiento del necesario.

Para utilizar el almacenamiento de modo más eficaz, establezca db2.jcc.charOutputSize con la longitud máxima prevista para cualquier parámetro INOUT u OUT de `Types.CHAR`.

db2.jcc.charOutputSize no tiene ningún efecto sobre los parámetros INOUT u OUT registrados como `Types.VARCHAR` o `Types.LONGVARCHAR`. El controlador utiliza la longitud por omisión, 32767, para los parámetros `Types.VARCHAR` y `Types.LONGVARCHAR`.

El valor que seleccione para db2.jcc.charOutputSize debe tener en cuenta la posibilidad de ampliación durante la conversión de caracteres. Puesto que IBM Data Server Driver para JDBC y SQLJ no tiene información sobre el CCSID correspondiente al servidor que se utiliza para los valores de los parámetros de salida, el controlador solicita los datos de salida del procedimiento almacenado en UTF-8 Unicode. El valor de db2.jcc.charOutputSize debe ser el número máximo de bytes que se necesitan después de que el valor del parámetro se convierta a UTF-8 Unicode. Los caracteres UTF-8 Unicode pueden necesitar hasta tres bytes. (El símbolo del euro es un ejemplo de carácter UTF-8 de tres bytes). Para garantizar que el valor de db2.jcc.charOutputSize sea suficientemente elevado, si no dispone de información sobre los datos de salida, establezca como valor de db2.jcc.charOutputSize un valor del triple de la longitud definida del parámetro CHAR más elevado.

Esta propiedad de configuración se aplica únicamente a DB2 para z/OS.

db2.jcc.currentSchema o db2.jcc.override.currentSchema
Especifica el nombre de esquema por omisión que se utiliza para calificar objetos de base de datos no calificados en sentencias de SQL preparadas dinámicamente. Este valor de la propiedad establece el valor del registro especial CURRENT SCHEMA en el servidor de bases de datos. El nombre de esquema es sensible a las mayúsculas y minúsculas y debe especificarse en caracteres en mayúsculas.

Esta propiedad de configuración se aplica únicamente a DB2 para z/OS o DB2 Database para Linux, UNIX y Windows.

db2.jcc.currentSQLID o db2.jcc.override.currentSQLID
Especifica:

- El ID de autorización que se utiliza para la comprobación de autorizaciones en las sentencias CREATE, GRANT y REVOKE de SQL preparadas dinámicamente.
- El propietario de un espacio de tablas, base de datos, grupo de almacenamiento o sinónimo que es creado por una sentencia CREATE emitida dinámicamente.
- El calificador implícito de todos los nombres de tabla, vista, alias e índice especificados en sentencias de SQL dinámico.

La propiedad currentSQLID define el valor del registro especial CURRENT SQLID en un servidor DB2 para z/OS. Si la propiedad currentSQLID no está definida, el nombre de esquema por omisión es el valor del registro especial CURRENT SQLID.

Esta propiedad de configuración se aplica únicamente a DB2 para z/OS.

db2.jcc.decimalRoundingMode o db2.jcc.override.decimalRoundingMode
Especifica la modalidad de redondeo para la asignación de variables de coma flotante decimal o columnas DECFLOAT en servidores de datos DB2 para z/OS o DB2 Database para Linux, UNIX y Windows.

Los valores posibles son:

com.ibm.db2.jcc.DB2BaseDataSource.ROUND_DOWN (1)
Redondea el valor hacia 0 (truncamiento). Los dígitos descartados no se tienen en cuenta.

com.ibm.db2.jcc.DB2BaseDataSource.ROUND_CEILING (2)
Redondea el valor hacia infinito positivo. Si todos los dígitos descartados son ceros, o si el signo es negativo, el resultado permanece inalterado salvo por la eliminación de los dígitos descartados. En otro caso, el coeficiente del resultado se incrementa en 1.

com.ibm.db2.jcc.DB2BaseDataSource.ROUND_HALF_EVEN (3)
Redondea el valor hasta el valor más cercano; si los valores son equidistantes, redondea el valor de forma que el dígito final sea par. Si los dígitos descartados representan un valor mayor que la mitad (0,5) del valor de un dígito en la posición izquierda siguiente, el coeficiente del resultado se incrementa en 1. Si éstos son inferiores a 0,5 el coeficiente del resultado no se ajustará (es decir, que los dígitos descartados se ignorarán). En otro caso, el coeficiente del resultado permanece inalterado si su dígito más a la derecha es par, o se incrementa en 1 si su dígito más a la derecha es impar (para convertirlo en un dígito par).

com.ibm.db2.jcc.DB2BaseDataSource.ROUND_HALF_UP (4)
Redondea el valor hasta el valor más cercano; si los valores son equidistantes, redondea el valor por exceso. Si los dígitos descartados representan un valor mayor o igual que la mitad (0,5) del valor del dígito en la posición izquierda siguiente, el coeficiente del resultado se incrementa en 1. En otro caso, los dígitos descartados no se tienen en cuenta.

com.ibm.db2.jcc.DB2BaseDataSource.ROUND_FLOOR (6)
Redondea el valor hacia infinito negativo. Si todos los dígitos descartados son ceros, o si el signo es positivo, el resultado permanece inalterado salvo por la eliminación de los dígitos descartados. En otro caso, el signo es negativo y el coeficiente del resultado se incrementa en 1.

com.ibm.db2.jcc.DB2BaseDataSource.ROUND_UNSET (-2147483647)
No se ha establecido explícitamente ninguna modalidad de redondeo. IBM Data Server Driver para JDBC y SQLJ no utiliza la propiedad decimalRoundingMode para establecer la modalidad de redondeo en el servidor de bases de datos. La modalidad de redondeo es ROUND_HALF_EVEN.

Si establece explícitamente el valor de db2.jcc.decimalRoundingMode o db2.jcc.override.decimalRoundingMode, ese valor actualiza el valor del registro especial CURRENT DECFLOAT ROUNDING MODE en un servidor de datos DB2 para z/OS.

Si se establece de forma explícita el valor de db2.jcc.decimalRoundingMode o db2.jcc.override.decimalRoundingMode, ese valor no actualiza el valor del registro especial CURRENT DECFLOAT ROUNDING MODE en un servidor de datos DB2 Database para Linux, UNIX y Windows. Si el valor en el que ha establecido db2.jcc.decimalRoundingMode o db2.jcc.override.decimalRoundingMode no es igual que el valor del registro especial CURRENT DECFLOAT ROUNDING MODE, se genera una Exception. Para cambiar el valor del servidor de datos, tiene que establecer ese valor con el parámetro de configuración de base de datos decflt_rounding.

decimalRoundingMode no afecta las asignaciones de valores decimales. IBM Data Server Driver para JDBC y SQLJ siempre redondea a la baja los valores decimales.

db2.jcc.defaultSQLState
Especifica el valor de SQLSTATE que IBM Data Server Driver para JDBC y SQLJ devuelve al cliente para objetos SQLException o SQLWarning que tienen valores de SQLSTATE nulos. Esta propiedad de configuración puede especificarse de las formas siguientes:

db2.jcc.defaultSQLState
Si se ha especificado db2.jcc.defaultSQLState sin ningún valor, IBM Data Server Driver para JDBC y SQLJ devuelve 'FFFFF'.

db2.jcc.defaultSQLState=*xxxxx*
xxxxx es el valor que IBM Data Server Driver para JDBC y SQLJ devuelve cuando el valor de SQLSTATE es nulo. Si *xxxxx* sobrepasa los cinco bytes, el controlador trunca el valor a cinco bytes. Si *xxxxx* tiene menos de cinco bytes, el controlador rellena *xxxxx* con blancos a la derecha.

Si no se ha especificado db2.jcc.defaultSQLState, IBM Data Server Driver para JDBC y SQLJ devuelve un valor de SQLSTATE nulo.

Esta propiedad de configuración se aplica únicamente a DB2 para z/OS.

db2.jcc.disableSQLJProfileCaching
Especifica si los perfiles serializados se colocan en antememoria cuando se restablece la JVM en la que se está ejecutando la aplicación correspondiente. db2.jcc.disableSQLJProfileCaching es aplicable solamente a aplicaciones que se ejecutan en una JVM reiniciable (aplicaciones que se ejecutan en un entorno de procedimiento almacenado CICS, IMS o Java), y utilizan IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS. Los valores posibles son:

YES Los perfiles serializados SQLJ no se colocan en antememoria cada vez que se restablece la JVM, de modo que las nuevas versiones de los perfiles serializados se cargan cuando la JVM se restablece. Utilice esta opción cuando una aplicación esté en fase de desarrollo y se creen nuevas versiones de la aplicación y sus perfiles serializados con frecuencia.

NO Los perfiles serializados SQLJ se colocan en antememoria cuando se restablece la JVM. NO es el valor por omisión.

Esta propiedad de configuración se aplica únicamente a DB2 para z/OS.

db2.jcc.dumpPool
Especifica los tipos de estadísticas que se escriben sobre los sucesos de la agrupación de transporte global, además de las estadísticas de resumen. La agrupación de transporte global se utiliza para el concentrador de conexiones y el equilibrado de la carga de trabajo Sysplex.

Las propiedades db2.jcc.dumpPoolStatisticsOnSchedule y db2.jcc.dumpPoolStatisticsOnScheduleFile también deben establecerse en modalidad de grabación de estadísticas antes de que se graben estadísticas.

Con la propiedad db2.jcc.dumpPool se pueden especificar uno o más de los tipos de estadísticas siguientes:
- DUMP_REMOVE_OBJECT (hexadecimal: X'01', decimal: 1)
- DUMP_GET_OBJECT (hexadecimal: X'02', decimal: 2)
- DUMP_WAIT_OBJECT (hexadecimal: X'04', decimal: 4)
- DUMP_SET_AVAILABLE_OBJECT (hexadecimal: X'08', decimal: 8)
- DUMP_CREATE_OBJECT (hexadecimal: X'10', decimal: 16)
- DUMP_SYSPLEX_MSG (hexadecimal: X'20', decimal: 32)
- DUMP_POOL_ERROR (hexadecimal: X'80', decimal: 128)

Para rastrear más de un tipo de suceso, añada los valores correspondientes a los tipos de sucesos que desee rastrear. Por ejemplo, suponga que desea rastrear los sucesos DUMP_GET_OBJECT y DUMP_CREATE_OBJECT. Los equivalentes numéricos de estos valores son 2 y 16, por lo que debe especificar 18 para el valor de db2.jcc.dumpPool.

El valor por omisión es 0, que significa que sólo se graban las estadísticas de resumen correspondientes a la agrupación de transporte global.

Esta propiedad de configuración se aplica únicamente a DB2 para z/OS o IBM Informix.

db2.jcc.dumpPoolStatisticsOnSchedule
Especifica la frecuencia, en segundos, con que las estadísticas de agrupación de transporte global se graban en el archivo especificado por db2.jcc.dumpPoolStatisticsOnScheduleFile. La agrupación de objetos de transporte global se utiliza para el concentrador de conexiones y el equilibrado de la carga de trabajo Sysplex.

El valor por omisión es -1. -1 significa que no se graban las estadísticas de agrupación de transporte global.

Esta propiedad de configuración se aplica únicamente a DB2 para z/OS o IBM Informix.

db2.jcc.dumpPoolStatisticsOnScheduleFile
Especifica el nombre del archivo en el que se escriben las estadísticas de la agrupación de transporte global. La agrupación de transporte global se utiliza para el concentrador de conexiones y el equilibrado de la carga de trabajo Sysplex.

Si no se especifica db2.jcc.dumpPoolStatisticsOnScheduleFile, no se graban las estadísticas de agrupación de transporte global.

Esta propiedad de configuración se aplica únicamente a DB2 para z/OS o IBM Informix.

db2.jcc.enableInetAddressGetHostName
Especifica si IBM Data Server Driver para JDBC y SQLJ utiliza los métodos InetAddress.getHostName y InetAddress.getCanonicalHostName para determinar el nombre de sistema principal para una dirección IP.

db2.jcc.enableInetAddressGetHostName solo se aplica a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4. Los valores posibles son:

true IBM Data Server Driver para JDBC y SQLJ utiliza los métodos InetAddress.getHostName y InetAddress.getCanonicalHostName para determinar el nombre de sistema principal para una dirección IP.

false IBM Data Server Driver para JDBC y SQLJ utiliza el método InetAddress.getHostAddress para determinar el nombre de sistema principal para una dirección IP. `false` es el valor por omisión.

db2.jcc.override.enableMultiRowInsertSupport
Especifica si IBM Data Server Driver para JDBC y SQLJ utiliza INSERT de varias filas para operaciones INSERT o MERGE por lotes, cuando el servidor de datos de destino es un servidor DB2 para z/OS que soporta las operaciones INSERT de varias filas. Las operaciones por lotes deben ser llamadas PreparedStatement con marcadores de parámetro. El valor por omisión es `true`.

db2.jcc.override.enableMultiRowInsertSupport debe establecerse en false si las sentencias INSERT FROM SELECT se ejecutan en un lote. En caso contrario, el controlador emite una BatchUpdateException.

Los valores posibles son:

true Especifica que IBM Data Server Driver para JDBC y SQLJ utiliza INSERT de varias filas para las operaciones INSERT o MERGE por lotes, cuando el servidor de datos de destino es un servidor DB2 para z/OS que soporta las operaciones INSERT de varias filas. Éste es el valor por omisión.

false Especifica que IBM Data Server Driver para JDBC y SQLJ no utiliza INSERT de varias filas para las operaciones INSERT o MERGE por lotes, cuando el servidor de datos de destino es un servidor DB2 para z/OS que soporta las operaciones INSERT de varias filas.

db2.jcc.encryptionAlgorithm o db2.jcc.override.encryptionAlgorithm
Especifica si el controlador IBM Data Server Driver para JDBC y SQLJ utiliza cifrado DES de 56 bits (débil) o cifrado AES de 256 bits (fuerte).

db2.jcc.encryptionAlgorithm o db2.jcc.override.encryptionAlgorithm solo pueden especificarse si db2.jcc.securityMechanism o db2.jcc.securityMechanism se han establecido en 7 ó 9.

Los valores posibles son:

1 El controlador utiliza cifrado DES de 56 bits.

2 El controlador utiliza cifrado AES de 256 bits, si este cifrado es compatible con el servidor de bases de datos. El cifrado EAS de 256 bits solo está disponible para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4.

Para el cifrado AES, necesita un archivo de política no restringida para JCE. Para IBM SDK para Java, el archivo está disponible en la ubicación siguiente:

`https://www.software.ibm.com/webapp/iwm/web/preLogin.do?source=jcesdk`

El SDK para Java de Oracle podría soportar el cifrado de AES, pero no el cifrado DES. Si se utiliza el cifrado de AES con el SDK para Java de Oracle, es necesario tener instalado el archivo JCE Unlimited Strength Jurisdiction Policy File. Este archivo está disponible en Oracle. Si no se

encuentra el archivo JCE Unlimited Strength Jurisdiction Policy File, se emite una java.security.InvalidKeyException.

db2.jcc.encryptionAlgorithm solo puede especificarse si el valor db2.jcc.securityMechanism, db2.jcc.override.securityMechanism o securityMechanism se ha establecido para la seguridad de contraseña cifrada o la seguridad de ID de usuario y contraseña cifrados.

db2.jcc.jmxEnabled
Especifica si Java Management Extensions (JMX) está habilitado para la instancia de IBM Data Server Driver para JDBC y SQLJ. JMX debe estar habilitado para que las aplicaciones puedan utilizar el controlador de rastreo remoto.

Los valores posibles son:

true o yes
Indica que JMX está habilitado.

Cualquier otro valor
Indica que JMX está inhabilitado. Éste es el valor por omisión.

db2.jcc.lobOutputSize
Especifica el número de bytes de almacenamiento que IBM Data Server Driver para JDBC y SQLJ necesita asignar para los valores LOB de salida si el controlador no puede determinar el tamaño de esos LOB. Esta situación se produce con los parámetros de salida de procedimiento almacenado de LOB. db2.jcc.lobOutputSize es aplicable solamente a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS.

El valor por omisión de db2.jcc.lobOutputSize es 1048576. En el caso de los sistemas con limitaciones de almacenamiento y LOB más pequeños, establezca el valor de db2.jcc.lobOutputSize en un número más bajo.

Por ejemplo, si sabe que el tamaño de LOB de salida es de 64000 como máximo, establezca db2.jcc.lobOutputSize en 64000.

Esta propiedad de configuración se aplica únicamente a DB2 para z/OS.

db2.jcc.maxConnCachedParamBufferSize
Especifica el tamaño máximo del almacenamiento intermedio interno que se utiliza para almacenar en la antememoria los valores de parámetros de entrada de los objetos PreparedStatement. El almacenamiento intermedio almacena valores en el lado del código nativo procedentes del lado del código Java del controlador de IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS. El almacenamiento intermedio es utilizado por todos los objetos PreparedStatement para Connection. El valor por omisión es 1048576 (1 MB). El valor por omisión debe ser adecuado para la mayoría de los usuarios. Establezca db2.jcc.maxConnCachedParamBufferSize en un valor mayor si muchas de las aplicaciones que se ejecutan bajo la instancia del controlador tienen objetos PreparedStatement con grandes cantidades de parámetros de entrada o parámetros de entrada grandes. El valor db2.jcc.maxConnCachedParamBufferSize debe ser mayor que el tamaño máximo de todos los datos de parámetros de entrada para Connection. Sin embargo, también debe tener en cuenta el número total de conexiones y la cantidad máxima de memoria que está disponible cuando se establece el valor db2.jcc.maxConnCachedParamBufferSize.

El almacenamiento intermedio existe mientras dure un objeto Connection, a menos que alcance el tamaño máximo especificado. Si esto sucede, el almacenamiento intermedio se libera en cada llamada al código nativo. El

almacenamiento intermedio correspondiente en el lado del código Java se libera en las llamadas PreparedStatement.clearParameters y PreparedStatement.close. Los almacenamientos intermedios no se borran si una aplicación llama a PreparedStatement.clearParameters y los almacenamientos intermedios no han alcanzado el tamaño máximo.

db2.jcc.maxRefreshInterval
Para el equilibrado de la carga de trabajo, especifica la cantidad máxima de tiempo en segundos entre las distintas actualizaciones de la copia de cliente de la lista de servidores. El valor mínimo válido es 1.

Para la Versión 3.63, 4.13 o posterior de IBM Data Server Driver para JDBC y SQLJ, el valor por omisión es 10 segundos. Para versiones anteriores del controlador, el valor por omisión es 30 segundos.

db2.jcc.maxTransportObjectIdleTime
Especifica la cantidad de tiempo en segundos que un objeto de transporte no utilizado debe permanecer en una agrupación de objetos de transporte global para poderlo suprimir de la agrupación. Los objetos de transporte se utilizan para el concentrador de conexiones y el equilibrado de la carga de trabajo Sysplex.

El valor por omisión para db2.jcc.maxTransportObjectIdleTime es 10. Si se establece db2.jcc.maxTransportObjectIdleTime en un valor menor que 0, los objetos de transporte no utilizados se suprimen de la agrupación inmediatamente. Esta acción **no** es recomendable, ya que puede provocar una importante disminución del rendimiento.

db2.jcc.maxTransportObjects
Especifica el límite superior del número de objetos de transporte de una agrupación de objetos de transporte global para el concentrador de conexión y el equilibrado de carga de trabajo Sysplex. Cuando el número de objetos de transporte de la agrupación alcanza el valor de db2.jcc.maxTransportObjects, los objetos de transporte que no se han utilizado durante un período de tiempo superior al valor de db2.jcc.maxTransportObjectIdleTime se suprimen de la agrupación.

Para la Versión 3.63, 4.13 o posterior de IBM Data Server Driver para JDBC y SQLJ, el valor por omisión es 1000. Para versiones anteriores del controlador, el valor por omisión es -1.

Todos los valores que sean 0 o que sean menores que 0 indican que no hay límite alguno en cuanto al número de objetos de transporte que puede haber en la agrupación de objetos de transporte global.

db2.jcc.maxTransportObjectWaitTime
Especifica la cantidad de tiempo en segundos que una aplicación espera un objeto de transporte si se alcanza el valor de db2.jcc.maxTransportObjects. Los objetos de transporte se utilizan para el concentrador de conexiones y el equilibrado de la carga de trabajo Sysplex. Cuando una aplicación espera más tiempo que el especificado por el valor de db2.jcc.maxTransportObjectWaitTime, la agrupación de objetos de transporte global emite una excepción de SQL (SQLException).

Los valores negativos significan que las aplicaciones esperan sin limitación de tiempo.

Para la Versión 3.63, 4.13 o posterior de IBM Data Server Driver para JDBC y SQLJ, el valor por omisión es 1 segundo. Para versiones anteriores del controlador, el valor por omisión es -1.

db2.jcc.minTransportObjects
Especifica el límite inferior del número de objetos de transporte de una agrupación de objetos de transporte global para el concentrador de conexiones y el equilibrado de carga de trabajo Sysplex. Cuando se crea una JVM, no hay objetos de transporte en la agrupación. Los objetos de transporte se añaden a la agrupación a medida que se necesitan. Una vez alcanzado el valor de db2.jcc.minTransportObjects, el número de objetos de transporte de la agrupación de objetos de transporte global nunca será inferior al valor de db2.jcc.minTransportObjects mientras exista esa JVM.

El valor por omisión de db2.jcc.minTransportObjects es 0. Todos los valores que sean 0 o que sean menores que 0 indican que no hay límite alguno en cuanto al número de objetos de transporte que puede haber en la agrupación de objetos de transporte global.

db2.jcc.outputDirectory
Especifica dónde almacena IBM Data Server Driver para JDBC y SQLJ los archivos de antememoria o de anotaciones cronológicas temporales.

Si esta propiedad está establecida, IBM Data Server Driver para JDBC y SQLJ almacena los archivos siguientes en el directorio especificado:

jccServerListCache.bin
Contiene una copia de la información sobre los servidores primario y alternativo para el redireccionamiento de cliente automático en un entorno DB2 pureScale.

Este archivo solo se aplica a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 para DB2 Database para Linux, UNIX y Windows.

Si no se especifica db2.jcc.outputDirectory, IBM Data Server Driver para JDBC y SQLJ busca un directorio especificado en la propiedad del sistema java.io.tmpdir. Si la propiedad del sistema java.io.tmpdir tampoco está especificada, el controlador utiliza solamente la antememoria almacenada en la memoria para la información del servidor primario y del servidor alternativo. Si se especifica un directorio, pero no se puede acceder a jccServerListCache.bin, el controlador utiliza solamente la antememoria almacenada en la memoria para la lista de servidores.

jccdiag.log
Contiene información de diagnóstico grabada por IBM Data Server Driver para JDBC y SQLJ.

Si no se especifica db2.jcc.outputDirectory, IBM Data Server Driver para JDBC y SQLJ busca un directorio especificado en la propiedad del sistema java.io.tmpdir. Si la propiedad del sistema java.io.tmpdir tampoco está especificada, el controlador no graba la información de diagnóstico en jccdiag.log. Si se especifica un directorio, pero no se puede acceder a jccdiag.log, el controlador no graba información de diagnóstico en jccdiag.log.

connlicj.bin
Contiene información sobre la verificación de licencia de IBM Data Server Driver para JDBC y SQLJ, para las conexiones directas con DB2 para z/OS. IBM Data Server Driver para JDBC y SQLJ graba este archivo cuando se realiza la verificación de licencia de servidor correctamente para un servidor de datos. Cuando se almacena en el

cliente una copia de la información de verificación de licencia, se puede mejorar el rendimiento de la verificación de la licencia en las conexiones siguientes.

Si no se especifica db2.jcc.outputDirectory, IBM Data Server Driver para JDBC y SQLJ busca un directorio especificado en la propiedad del sistema java.io.tmpdir. Si la propiedad del sistema java.io.tmpdir tampoco está especificada, el controlador no almacena una copia de la información de verificación de licencia de servidor en el cliente. Si se especifica un directorio, pero no se puede acceder a connlicj.bin, el controlador no almacena en el cliente una copia de la información de verificación de licencia de servidor.

IBM Data Server Driver para JDBC y SQLJ no crea el directorio. Debe crear el directorio y asignar los permisos de archivo necesarios.

db2.jcc.outputDirectory puede especificar una vía de acceso absoluta o relativa. Sin embargo, se recomienda una vía de acceso absoluta.

db2.jcc.pkList
Especifica una lista de paquetes que se utiliza para la llamada subyacente CREATE THREAD de RRSAF cuando se establece una conexión JDBC o SQLJ con una fuente de datos. Especifique esta propiedad si no vincula planes para sus programas SQLJ ni para el controlador JDBC. Si especifica esta propiedad, **no especifique db2.jcc.planName.**

db2.jcc.pkList es aplicable solamente a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS. db2.jcc.pkList no es aplicable a las aplicaciones que se ejecutan en CICS o IMS ni a los procedimientos almacenados Java. El controlador JDBC pasa por alto el valor db2.jcc.pkList en esos casos.

Recomendación: utilice db2.jcc.pkList en lugar de db2.jcc.planName.

El formato de la lista de paquetes es:

El valor por omisión de db2.jcc.pkList es NULLID.*.

Si especifica el parámetro -collection al ejecutar `com.ibm.db2.jcc.DB2Binder`, el ID de colección que especifique para los paquetes de IBM Data Server Driver para JDBC y SQLJ al ejecutar `com.ibm.db2.jcc.DB2Binder` también debe estar en la lista de paquetes de la propiedad db2.jcc.pkList.

Puede alterar temporalmente db2.jcc.pkList estableciendo la propiedad `pkList` para un objeto Connection o DataSource.

En el ejemplo siguiente se especifica una lista de paquetes para una instancia de IBM Data Server Driver para JDBC y SQLJ cuyos paquetes están en la colección JDBCCID. Las aplicaciones SQLJ preparadas en esta instancia de controlador están vinculadas a las colecciones SQLJCID1, SQLJCID2 o SQLJCID3.

```
db2.jcc.pkList=JDBCCID.*,SQLJCID1.*,SQLJCID2.*,SQLJCID3.*
```

Esta propiedad de configuración se aplica únicamente a DB2 para z/OS.

db2.jcc.planName
Especifica un nombre de plan de DB2 para z/OS que es utilizado para la

llamada subyacente a CREATE THREAD de RRSAF cuando se establece una conexión JDBC o SQLJ con una fuente de datos. Especifique esta propiedad si no vincula planes para sus programas SQLJ ni para los paquetes del controlador JDBC. Si especifica esta propiedad, **no especifique db2.jcc.pkList.**

db2.jcc.planName es aplicable solamente a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS. db2.jcc.planName no es aplicable a las aplicaciones que se ejecutan en CICS o IMS ni a los procedimientos almacenados Java. El controlador JDBC pasa por alto el valor db2.jcc.planName en esos casos.

Si no especifica esta propiedad o la propiedad db2.jcc.pkList, IBM Data Server Driver para JDBC y SQLJ utiliza el valor por omisión de db2.jcc.pkList, que es NULLID.*.

Si especifica db2.jcc.planName, debe vincular los paquetes que genere al ejecutar `com.ibm.db2.jcc.DB2Binder` a un plan cuyo nombre sea el valor de esta propiedad. También debe vincular todos los paquetes SQLJ a un plan cuyo nombre sea el valor de esta propiedad.

Puede alterar temporalmente db2.jcc.planName estableciendo la propiedad `planName` para un objeto Connection o DataSource.

En el ejemplo siguiente se especifica el nombre de plan MYPLAN para los paquetes de JDBC y SQLJ de IBM Data Server Driver para JDBC y SQLJ.

```
db2.jcc.planName=MYPLAN
```

Esta propiedad de configuración se aplica únicamente a DB2 para z/OS.

db2.jcc.progressiveStreaming o db2.jcc.override.progressiveStreaming
Especifica si el controlador JDBC utiliza la modalidad continua progresiva cuando esta función es compatible con la fuente de datos.

Cuando se utiliza la modalidad continua progresiva, también conocida como formato de datos dinámico, la fuente de datos determina dinámicamente la forma más eficiente de devolver datos LOB o XML, de acuerdo con el tamaño de los objetos LOB o XML.

Los valores válidos son:

1 Utilizar la modalidad continua progresiva si la fuente de datos es compatible con ésta.

2 No utilizar la modalidad continua progresiva.

db2.jcc.rollbackOnShutdown
Especifica si DB2 para z/OS fuerza una operación de retrotracción e inhabilita operaciones subsiguientes de una unidad de trabajo en conexiones JDBC durante el proceso de cierre de JVM.

db2.jcc.rollbackOnShutdown es aplicable solamente a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2.

db2.jcc.rollbackOnShutdown no es aplicable a entornos de CICS, IMS, procedimiento almacenado ni WebSphere Application Server.

Los valores posibles son:

yes o true
IBM Data Server Driver para JDBC y SQLJ hace que DB2 para z/OS fuerce una operación de retrotracción e inhabilita operaciones subsiguientes de una unidad de trabajo en conexiones JDBC durante el proceso de cierre de JVM.

Cualquier otro valor
IBM Data Server Driver para JDBC y SQLJ no emprende ninguna acción con respecto al proceso de retrotracciones durante el proceso de cierre de JVM. Éste es el valor por omisión.

Esta propiedad de configuración se aplica únicamente a DB2 para z/OS.

db2.jcc.securityMechanism or db2.jcc.override.securityMechanism
Especifica el mecanismo de seguridad de DRDA. Los valores posibles son:

3 ID de usuario y contraseña

4 ID de usuario solamente

7 ID de usuario, contraseña cifrada

9 ID de usuario y contraseña cifrados

11 Kerberos. Este valor no se aplica a las conexiones con IBM Informix.

12 ID de usuario y datos confidenciales cifrados. Este valor es aplicable solamente a las conexiones con DB2 para z/OS.

13 ID de usuario y contraseña cifrados, y datos confidenciales cifrados. Este valor no se aplica a las conexiones con IBM Informix.

15 Seguridad por plugin. Este valor es aplicable solamente a las conexiones con DB2 Database para Linux, UNIX y Windows.

16 ID de usuario cifrado. Este valor no se aplica a las conexiones con IBM Informix.

18 Seguridad de certificado de cliente mediante SSL. Este valor es aplicable solamente a las conexiones con DB2 para z/OS Versión 10 y posteriores.

El mecanismo de seguridad que se especifica mediante esta propiedad es el único mecanismo que se utiliza. Si el mecanismo de seguridad no está soportado por la conexión, se emite una excepción.

El valor por omisión para db2.jcc.securityMechanism es 3. Si el servidor no da soporte a la seguridad de ID de usuario y contraseña, pero sí da soporte a la seguridad de ID de usuario y contraseña cifrados (9), el controlador IBM Data Server Driver para JDBC y SQLJ actualiza el mecanismo de seguridad por la seguridad de ID de usuario y contraseña cifrados e intenta establecer la conexión con el servidor. Cualquier otra discrepancia en el soporte del mecanismo de seguridad entre el solicitante y el servidor da como resultado un error.

Esa propiedad no se aplica a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS.

db2.jcc.sendCharInputsUTF8
Especifica si IBM Data Server Driver para JDBC y SQLJ convierte los datos de entrada de tipo carácter al CCSID del servidor de bases de datos DB2 para z/OS, o si envía los datos en la codificación UTF-8 para su conversión por el servidor de bases de datos. La propiedad db2.jcc.sendCharInputsUTF8 es aplicable solamente a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 con servidores de bases de datos DB2 para z/OS. Si esta propiedad también está establecida a nivel de conexión, el valor a nivel de conexión prevalece sobre este valor.

Los valores posibles son:

no, false o 2

Especifica que IBM Data Server Driver para JDBC y SQLJ convierte los datos de entrada de tipo carácter a la codificación de destino antes de enviar los datos al servidor de bases de datos DB2 para z/OS. Éste es el valor por omisión.

yes, true o 1

Especifica que IBM Data Server Driver para JDBC y SQLJ envía los datos de entrada de tipo carácter al servidor de bases de datos DB2 para z/OS en la codificación UTF-8. La fuente de datos convierte los datos desde la codificación UTF-8 al CCSID de destino.

Especifique yes, true o 1 solamente si la conversión al CCSID de destino realizada por el SDK de Java produce problemas de conversión de caracteres. El problema más habitual se produce cuando se utiliza IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 para insertar un carácter Unicode de salto de línea (U+000A) en una columna de tabla cuyo CCSID es 37, y luego se recuperan esos datos a partir de un cliente que no es un cliente z/OS. Si el SDK de Java realiza la conversión durante la inserción del carácter en la columna, el carácter de salto de línea se convierte en el carácter de línea nueva de EBCDIC, X'15'. Pero, durante la recuperación de los datos, algunos SDK de Java que se ejecutan en sistemas operativos distintos de z/OS convierten el carácter X'15' al carácter de línea siguiente de Unicode (U+0085), en lugar de convertirlo al carácter de salto de línea (U+000A). El carácter de línea siguiente produce un comportamiento inesperado en algunos analizadores sintácticos de XML. Si establece db2.jcc.sendCharInputsUTF8 en yes, el servidor de bases de datos DB2 para z/OS convierte el carácter U+000A al carácter de salto de línea de EBCDIC (X'25') durante la inserción en la columna, por lo que el carácter se recupera siempre como carácter de salto de línea.

La conversión de datos al CCSID de destino en la fuente de datos puede hacer que IBM Data Server Driver para JDBC y SQLJ utilice más memoria que la conversión por el controlador. El controlador asigna memoria para la conversión de datos de tipo carácter desde la codificación de origen a la codificación de los datos que envía a la fuente de datos. La cantidad de espacio que el controlador asigna para datos de tipo carácter que se envían a una columna de tabla está basada en la longitud máxima posible de los datos. Los datos con codificación UTF-8 pueden necesitar hasta tres bytes para cada carácter. Por tanto, si el controlador envía datos UTF-8 a la fuente de datos, el controlador necesita asignar un espacio tres veces mayor que el número máximo de caracteres de los datos de entrada. Si el controlador realiza la conversión y el CCSID de destino es un CCSID de un solo byte, el controlador necesita asignar solamente un espacio igual al número máximo de caracteres de los datos de entrada.

Por ejemplo, cualquiera de los valores siguientes de db2.jcc.sendCharInputsUTF8 hace que IBM Data Server Driver para JDBC y SQLJ convierta las series de caracteres de entrada a UTF-8, en lugar de la codificación de destino, antes de enviar los datos a la fuente de datos:

```
db2.jcc.sendCharInputsUTF8=yes
db2.jcc.sendCharInputsUTF8=true
db2.jcc.sendCharInputsUTF8=1
```

Esta propiedad de configuración se aplica únicamente a DB2 para z/OS.

db2.jcc.sqljToolsExitJVMOnCompletion
Especifica si los programas Java que contienen herramientas SQLJ como db2sqljcustomize y db2sqljbind emiten la llamada System.exit como respuesta a los programas emisores de llamadas.

Los valores posibles son:

true Especifica que los programas Java que contienen herramientas SQLJ emitan la llamada System.exit al completarse el proceso. `true` es el valor por omisión.

false Especifica que los programas Java que contienen herramientas SQLJ subyacentes no emiten la llamada System.exit.

db2.jcc.sqljUncustomizedWarningOrException
Especifica la acción que IBM Data Server Driver para JDBC y SQLJ llevará a cabo cuando se ejecute una aplicación SQLJ no personalizada. db2.jcc.sqljUncustomizedWarningOrException puede tener los valores siguientes:

0 IBM Data Server Driver para JDBC y SQLJ no emitirá un Warning o una Exception cuando se ejecute una aplicación SQLJ no personalizada. Éste es el valor por omisión.

1 IBM Data Server Driver para JDBC y SQLJ emitirá un Warning cuando se ejecute una aplicación SQLJ no personalizada.

2 IBM Data Server Driver para JDBC y SQLJ emitirá una Exception cuando se ejecute una aplicación SQLJ no personalizada.

Esta propiedad de configuración se aplica únicamente a DB2 para z/OS o DB2 Database para Linux, UNIX y Windows.

db2.jcc.traceDirectory o db2.jcc.override.traceDirectory
Habilita el rastreo de IBM Data Server Driver para JDBC y SQLJ para el código de controlador Java y especifica el directorio en el que se grabará la información de rastreo. Cuando se especifica db2.jcc.override.traceDirectory, la información de rastreo correspondiente a varias conexiones en el mismo DataSource se graba en varios archivos.

Cuando se especifica db2.jcc.override.traceDirectory, el resultado del rastreo de una conexión se guarda en un archivo denominado *nombre_archivo_origen_n*.

- *n* es la conexión número *n* correspondiente a una DataSource.
- Si no se especifica db2.jcc.traceFileName ni db2.jcc.override.traceFileName, *nombre-archivo* es traceFile. Si se especifica db2.jcc.traceFileName o db2.jcc.override.traceFileName, *nombre-archivo* es el valor de db2.jcc.traceFileName o db2.jcc.override.traceFileName.
- *origen* indica el origen del grabador de anotaciones cronológicas que se está utilizando. Los valores posibles de *origen* son:

  cpds Grabador de anotaciones cronológicas para un objeto DB2ConnectionPoolDataSource.

  driver Grabador de anotaciones cronológicas para un objeto DB2Driver.

  global Grabador de anotaciones cronológicas para un objeto DB2TraceManager.

  sds Grabador de anotaciones cronológicas para un objeto DB2SimpleDataSource.

  xads Grabador de anotaciones cronológicas para un objeto DB2XADataSource.

La propiedad db2.jcc.override.traceDirectory prevalece sobre la propiedad `traceDirectory` en el caso de un objeto Connection o DataSource.

Por ejemplo, si se especifica el valor siguiente para db2.jcc.override.traceDirectory, se habilita el rastreo del código Java de IBM Data Server Driver para JDBC y SQLJ para archivos del directorio /SYSTEM/tmp:

```
db2.jcc.override.traceDirectory=/SYSTEM/tmp
```

Debe establecer las propiedades del rastreo bajo la dirección del soporte de software de IBM.

db2.jcc.traceLevel o db2.jcc.override.traceLevel
Especifica qué se debe rastrear.

La propiedad db2.jcc.override.traceLevel prevalece sobre la propiedad `traceLevel` de un objeto Connection o DataSource.

Existe la posibilidad de especificar uno o más niveles de rastreo especificando un valor decimal. Los niveles de rastreo corresponden a los mismos niveles de rastreo que los definidos para la propiedad traceLevel en un objeto Connection o DataSource.

Si desea especificar más de un nivel de rastreo, utilice la operación (`|`) OR con los valores, y especifique el resultado con un valor decimal en la especificación db2.jcc.traceLevel o db2.jcc.override.traceLevel.

Por ejemplo, suponga que desea especificar TRACE_DRDA_FLOWS y TRACE_CONNECTIONS para db2.jcc.override.traceLevel. TRACE_DRDA_FLOWS tiene un valor hexadecimal de X'40'. TRACE_CONNECTION_CALLS tiene un valor hexadecimal de X'01'. Para especificar ambos niveles de rastreo, hay que realizar una operación OR a nivel de bits con los dos valores, con lo que se obtiene el valor X'41'. El valor decimal equivalente es 65, por lo tanto, especificaría:

```
db2.jcc.override.traceLevel=65
```

db2.jcc.ssid
Especifica el subsistema DB2 para z/OS con el que las aplicaciones realizan conexiones con IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS.

El valor db2.jcc.ssid puede ser el nombre del subsistema DB2 local, un nombre de enlace de grupos o un nombre de enlace de subgrupos.

Por ejemplo:

```
db2.jcc.ssid=DB2A
```

La propiedad Connection y DataSource de ssid alteran temporalmente db2.jcc.ssid.

Si especifica un nombre de enlace de grupos o de subgrupos y falla el subsistema DB2 que está conectado a una aplicación, la conexión termina. No obstante, cuando nuevas conexiones utilizan dicho nombre de enlace de grupos o de subgrupos, DB2 para z/OS utiliza un proceso de enlace de grupos o de subgrupos para buscar un subsistema DB2 activo al que conectarse.

Si no especifica la propiedad db2.jcc.ssid, IBM Data Server Driver para JDBC y SQLJ utiliza el valor de SSID del módulo de carga de valores por omisión de aplicación. Cuando se instala DB2 para z/OS, se crea un módulo de carga de valores por omisión de aplicación en el archivo *prefijo*.SDSNEXIT y en el

archivo *prefijo*.SDSNLOAD. Pueden crearse otros módulos de carga de valores por omisión de aplicación en otros conjuntos de datos para determinadas aplicaciones.

IBM Data Server Driver para JDBC y SQLJ debe cargar un módulo de carga de valores por omisión de aplicación para poder leer el valor SSID. z/OS busca el módulo de carga de valores por omisión de aplicación en los archivos de las ubicaciones siguientes, en el orden indicado:

1. JPA (área de paquete de trabajo)
2. TASKLIB
3. STEPLIB o JOBLIB
4. LPA
5. Bibliotecas de la lista de enlaces

Debe asegurarse de que, si el sistema tiene más de una copia del módulo de carga de valores por omisión de aplicación, z/OS primero encontrará el conjunto de datos que contiene la copia correcta para IBM Data Server Driver para JDBC y SQLJ.

Esta propiedad de configuración se aplica únicamente a DB2 para z/OS.

db2.jcc.traceFile o db2.jcc.override.traceFile
Habilita el rastreo de IBM Data Server Driver para JDBC y SQLJ para el código de controlador Java y especifica el nombre en el que se basan los nombres de archivo de rastreo.

Especifique un nombre de archivo de z/OS UNIX System Services totalmente calificado para el valor de la propiedad db2.jcc.override.traceFile.

La propiedad db2.jcc.override.traceFile prevalece sobre la propiedad `traceFile` en el caso de un objeto Connection o DataSource.

Por ejemplo, si se especifica el valor siguiente para db2.jcc.override.traceFile, se habilita el rastreo del código Java de IBM Data Server Driver para JDBC y SQLJ para un archivo denominado /SYSTEM/tmp/jdbctrace:

```
db2.jcc.override.traceFile=/SYSTEM/tmp/jdbctrace
```

Debe establecer las propiedades del rastreo bajo la dirección del soporte de software de IBM.

db2.jcc.traceFileAppend o db2.jcc.override.traceFileAppend
Especifica si se debe o no se debe añadir o sobregrabar los datos en el archivo especificado por la propiedad db2.jcc.override.traceFile. Los valores válidos son `true` o `false`. El valor por omisión es `false`, que significa que se sobrescribe el archivo especificado por la propiedad traceFile.

La propiedad db2.jcc.override.traceFileAppend prevalece sobre la propiedad `traceFileAppend` en el caso de un objeto Connection o DataSource.

Por ejemplo, si se especifica el valor siguiente para db2.jcc.override.traceFileAppend, los datos de rastreo se añaden al archivo de rastreo existente:

```
db2.jcc.override.traceFileAppend=true
```

Debe establecer las propiedades del rastreo bajo la dirección del soporte de software de IBM.

db2.jcc.traceFileCount
Especifica el número máximo de archivos de rastreo para el rastreo circular. IBM Data Server Driver para JDBC y SQLJ solo utiliza esta propiedad cuando db2.jcc.traceOption se establece en 1. El valor por omisión es 2.

Esa propiedad no se aplica a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS.

Debe establecer las propiedades del rastreo bajo la dirección del soporte de software de IBM.

db2.jcc.traceFileSize
Especifica el tamaño máximo de cada archivo de rastreo para el rastreo circular. IBM Data Server Driver para JDBC y SQLJ solo utiliza esta propiedad cuando db2.jcc.traceOption se establece en 1. El valor por omisión es 10485760 (10 MB).

Esa propiedad no se aplica a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS.

Debe establecer las propiedades del rastreo bajo la dirección del soporte de software de IBM.

db2.jcc.traceOption
Especifica el modo como se recopilan los datos de rastreo. El tipo de datos de esta propiedad es int. Los valores posibles son:

0 Especifica que se genera un solo archivo de rastreo y que no existe ningún límite para el tamaño del archivo. Éste es el valor por omisión.

1 Especifica que IBM Data Server Driver para JDBC y SQLJ realiza un rastreo circular. El rastreo circular se realiza de la forma siguiente:

1. Cuando una aplicación graba su primer registro de rastreo, el controlador crea un archivo.
2. El controlador graba los datos de rastreo en el archivo.
3. Cuando el tamaño del archivo es igual al valor de la propiedad db2.jcc.traceFileSize, el controlador crea otro archivo.
4. El controlador repite los pasos 2 y 3 hasta que el número de archivos en los que se han grabado datos es igual al valor de la propiedad db2.jcc.traceFileCount.
5. El controlador graba datos en el primer archivo de rastreo y sobrescribe los datos existentes.
6. El controlador repite los pasos de 3 a 5 hasta que la aplicación finaliza.

Los nombres de archivo de los archivos de rastreo son los nombres de archivo que la propiedad db2.jcc.traceFile, db2.jcc.override.traceFile, db2.jcc.traceDirectory o db2.jcc.override.traceDirectory determina, con la adición de .1 para el primer archivo, .2 para el segundo archivo y así sucesivamente.

Esa propiedad no se aplica a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS.

Debe establecer las propiedades del rastreo bajo la dirección del soporte de software de IBM.

db2.jcc.tracePolling
Indica si IBM Data Server Driver para JDBC y SQLJ sondea el archivo de configuración para ver si hay cambios en directivas de rastreo y modifica el comportamiento de rastreo para que coincidan con las directivas nuevas. Los valores posibles son true (verdadero) o false (falso). False (falso) es el valor por omisión.

IBM Data Server Driver para JDBC y SQLJ modifica el comportamiento de rastreo al principio del siguiente intervalo de sondeo, tras modificar el archivo de propiedades de configuración. Si se establece db2.jcc.tracePolling en `true` mientras se está ejecutando una aplicación, se activa el rastreo y se vuelca al destino del rastreo la información sobre todos los objetos PreparedStatement creados por la aplicación antes del rastreo.

db2.jcc.tracePolling sondea las siguientes propiedades de configuración globales:
- db2.jcc.override.traceLevel
- db2.jcc.override.traceFile
- db2.jcc.override.traceDirectory
- db2.jcc.override.traceFileAppend

db2.jcc.tracePollingInterval
Especifica el intervalo, en segundos, para realizar el sondeo del archivo de configuración global de IBM Data Server Driver para JDBC y SQLJ para comprobar si hay cambios en las directivas de rastreo. El valor de la propiedad es un entero positivo. El valor por omisión es 60. Para utilizar el intervalo de sondeo de rastreo especificado, debe establecer la propiedad db2.jcc.tracePollingInterval *antes* de cargar e inicializar el controlador. Los cambios que se realicen en db2.jcc.tracePollingInterval tras cargar e inicializar el controlador no entrarán en vigor.

db2.jcc.t2zosTraceFile
Habilita el rastreo de IBM Data Server Driver para JDBC y SQLJ para el código de controlador nativo C/C++ para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 y especifica el nombre en el que se basan los nombres de archivo de rastreo. Esta propiedad es necesaria para recopilar datos de rastreo para el código de controlador nativo C/C++.

Especifique un nombre de archivo de z/OS UNIX System Services totalmente calificado para el valor de la propiedad db2.jcct.t2zosTraceFile.

Por ejemplo, si se especifica el valor siguiente para db2.jcct.t2zosTraceFile, se habilita el rastreo del código nativo C/C++ de IBM Data Server Driver para JDBC y SQLJ que se guardará en un archivo denominado /SYSTEM/tmp/jdbctraceNative:

```
db2.jcc.t2zosTraceFile=/SYSTEM/tmp/jdbctraceNative
```

Debe establecer las propiedades del rastreo bajo la dirección del soporte de software de IBM.

Esta propiedad de configuración se aplica únicamente a DB2 para z/OS.

db2.jcc.t2zosTraceBufferSize
Especifica el tamaño, en kilobytes, de un almacenamiento intermedio de rastreo en el almacenamiento virtual que se utiliza para rastrear el proceso realizado por el código de controlador nativo C/C++. Este valor es también la cantidad máxima de información de rastreo del controlador nativo C/C++ que se puede recopilar.

Especifique un valor entero entre 64 (64 KB) y 4096 (4096 KB). El valor por omisión es 256 (256 KB).

El controlador JDBC determina el tamaño de almacenamiento intermedio de rastreo como se muestra en la tabla siguiente:

Valor especificado (*n*)	Tamaño de almacenamiento intermedio de rastreo (KB)
<64	64
64<=*n*<128	64
128<=*n*<256	128
256<=*n*<512	256
512<=*n*<1024	512
1024<=*n*<2048	1024
2048<=*n*<4096	2048
n>=4096	4096

db2.jcc.t2zosTraceBufferSize se utiliza únicamente si la propiedad db2.jcc.t2zosTraceFile está establecida.

Recomendación: para evitar que el rendimiento se vea afectado, especifique un valor de 1024 o un valor inferior.

Por ejemplo, para establecer un tamaño de almacenamiento intermedio de rastreo de 1024 KB, utilice este valor:

```
db2.jcc.t2zosTraceBufferSize=1024
```

Debe establecer las propiedades del rastreo bajo la dirección del soporte de software de IBM.

Esta propiedad de configuración se aplica únicamente a DB2 para z/OS.

db2.jcc.t2zosTraceWrap
Habilita o inhabilita el reinicio del rastreo SQLJ. db2.jcc.t2zosTraceWrap puede tener uno de los valores siguientes:

1 Reiniciar el rastreo

0 No reiniciar el rastreo

El valor por omisión es 1. Este parámetro es opcional. Por ejemplo:

```
DB2SQLJ_TRACE_WRAP=0
```

Debe establecer db2.jcc.t2zosTraceWrap bajo la dirección del soporte de software de IBM.

Esta propiedad de configuración se aplica únicamente a DB2 para z/OS.

db2.jcc.useCcsid420ShapedConverter
Especifica si los caracteres arábigos incluidos en el CCSID 420 de EBCDIC se correlacionan con la codificación Cp420S.

db2.jcc.useCcsid420ShapedConverter es aplicable solamente a conexiones con servidores de bases de datos DB2 para z/OS.

Si el valor de db2.jcc.useCcsid420ShapedConverter es `true`, el CCSID 420 se correlaciona con la codificación Cp420S. Si el valor de db2.jcc.useCcsid420ShapedConverter es `false`, el CCSID 420 se correlaciona con la codificación Cp420. `false` es el valor por omisión.

Esta propiedad de configuración se aplica únicamente a DB2 para z/OS.

Soporte de controladores para las API de JDBC

Los controladores JDBC que pueden ser utilizados por sistemas de bases de datos DB2 e IBM Informix tienen niveles diferentes de compatibilidad para los métodos JDBC.

Las tablas siguientes muestran las interfaces de JDBC y los controladores soportados para cada una. Los controladores y sus plataformas soportadas son:

Tabla 57. Controladores JDBC para sistemas de base de datos DB2 e IBM Informix

Nombre del controlador JDBC	Fuente de datos asociada
IBM Data Server Driver para JDBC y SQLJ	DB2 Database para Linux, UNIX y Windows, DB2 para z/OS o IBM Informix
Controlador JDBC de IBM Informix (controlador IBM Informix JDBC)	IBM Informix

Si un método tiene formatos de JDBC 2.0 y JDBC 3.0, IBM Data Server Driver para JDBC y SQLJ es compatible con todos los formatos. El controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows solamente es compatible con los formatos JDBC 2.0.

Tabla 58. Soporte para métodos java.sql.Array

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ1	Soporte del controlador JDBC de IBM Informix
free[2]	Sí	No
getArray	Sí	Sí
getBaseType	Sí	Sí
getBaseTypeName	Sí	Sí
getResultSet	Sí	Sí

Notas:

1. Cuando se utiliza IBM Data Server Driver para JDBC y SQLJ, los métodos `Array` se pueden utilizar solamente para conexiones con fuentes de datos DB2 Database para Linux, UNIX y Windows.
2. Esto es un método de JDBC 4.0.

Tabla 59. Soporte para métodos java.sql.BatchUpdateException

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
Métodos heredados de java.lang.Exception	Sí	Sí
getUpdateCounts	Sí	Sí

Tabla 60. Soporte para métodos java.sql.Blob

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
free[1]	Sí	No
getBinaryStream	Sí[2]	Sí
getBytes	Sí	Sí
longitud	Sí	Sí
posición	Sí	Sí
setBinaryStream[3]	Sí	No

Tabla 60. Soporte para métodos java.sql.Blob (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
setBytes[3]	Sí	No
truncate[3]	Sí	No

Notas:

1. Esto es un método de JDBC 4.0.
2. Los formatos válidos de este método incluyen el siguiente formato de JDBC 4.0:
   `getBinaryStream(long pos, long longitud)`
3. Para versiones de IBM Data Server Driver para JDBC y SQLJ anteriores a la versión 3.50, estos métodos no se pueden utilizar si se pasa un blob a un procedimiento almacenado como un parámetro IN o INOUT y los métodos se utilizan en el blob del procedimiento almacenado.

Tabla 61. Soporte para métodos java.sql.CallableStatement

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
Métodos heredados de java.sql.Statement	Sí	Sí
Métodos heredados de java.sql.PreparedStatement	Sí[1]	Sí
getArray	No	No
getBigDecimal	Sí[3]	Sí
getBlob	Sí[3]	Sí
getBoolean	Sí[3]	Sí
getByte	Sí[3]	Sí
getBytes	Sí[3]	Sí
getClob	Sí[3]	Sí
getDate	Sí[3,5]	Sí
getDouble	Sí[3]	Sí
getFloat	Sí[3]	Sí
getInt	Sí[3]	Sí
getLong	Sí[3]	Sí
getObject	Sí[3,4,6]	Sí
getRef	No	No
getRowId[2]	Sí	No
getShort	Sí[3]	Sí
getString	Sí[3]	Sí
getTime	Sí[3,5]	Sí
getTimestamp	Sí[3,5]	Sí
getURL	Sí	No
registerOutParameter	Sí[7]	Sí[7]
setAsciiStream	Sí[8]	Sí
setBigDecimal	Sí[8]	Sí
setBinaryStream	Sí[8]	Sí
setBoolean	Sí[8]	Sí
setByte	Sí[8]	Sí

Tabla 61. Soporte para métodos java.sql.CallableStatement (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
setBytes	Sí[8]	Sí
setCharacterStream	Sí[8]	Sí
setDate	Sí[8]	Sí
setDouble	Sí[8]	Sí
setFloat	Sí[8]	Sí
setInt	Sí[8]	Sí
setLong	Sí[8]	Sí
setNull	Sí[8,9]	Sí
setObject	Sí[8,]	Sí
setShort	Sí[8]	Sí
setString	Sí[8]	Sí
setTime	Sí[8]	Sí
setTimestamp	Sí[8]	Sí
setURL	Sí	No
wasNull	Sí	Sí

Notas:

1. El método heredado getParameterMetaData no se puede utilizar si la fuente de datos es DB2 para z/OS.
2. Esto es un método de JDBC 4.0.
3. Los formatos siguientes de los métodos CallableStatement.getXXX no se pueden utilizar si la fuente de datos es DB2 para z/OS:

   `get`*`XXX`*`(String` *`Nombre del parámetro`*`)`
4. Recibe soporte el siguiente método de JDBC 4.1:

   `getObject(int` *`índiceParámetros`*`, java.lang.Class<T>` *`tipo`*`)`
   `getObject(java.lang.String` *`nombreParámetro`*`, java.lang.Class<T>` *`tipo`*`)`
5. El servidor de bases de datos no realiza ajustes de zona horaria para valores de fecha y hora. El controlador JDBC ajusta un valor para la zona horaria local después de recuperar el valor a partir del servidor si el usuario especifica un formato del método getDate, getTime o getTimestamp que incluye un parámetro java.util.Calendar.
6. El formato siguiente del método getObject no se soporta:

   `getObject(int` *`Índiceparámetros`*`, java.util.Map`
   `map``)`
7. El formato siguiente del método registerOutParameter no se soporta:

   `registerOutParameter(int` *`índiceParámetro`*`, int` *`tipoJdbc`*`, String` *`nombreTipo`*`)`
8. Los formatos siguientes de los métodos CallableStatement.setXXX no se pueden utilizar si la fuente de datos es DB2 para z/OS:

   `set`*`XXX`*`(String` *`Nombre del parámetro`*`,...)`
9. El formato siguiente de setNull no se soporta:

   `setNull(int` *`Índiceparámetros`*`, int` *`TipoJdbc`*`, String` *`nombreTipo`*`)`

Tabla 62. Soporte para métodos java.sql.Clob

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
free[1]	Sí	No

Tabla 62. Soporte para métodos java.sql.Clob (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
getAsciiStream	Sí	Sí
getCharacterStream	Sí2	Sí
getSubString	Sí	Sí
longitud	Sí	Sí
posición	Sí	Sí
setAsciiStream[3]	Sí	Sí
setCharacterStream[3]	Sí	Sí
setString[3]	Sí	Sí
truncate[3]	Sí	Sí

Notas:

1. Esto es un método de JDBC 4.0.
2. Los formatos válidos de este método incluyen el siguiente formato de JDBC 4.0:
   `getCharacterStream(long pos, long longitud)`
3. Para versiones de IBM Data Server Driver para JDBC y SQLJ anteriores a la versión 3.50, estos métodos no se pueden utilizar si se pasa un clob a un procedimiento almacenado como un parámetro IN o INOUT y los métodos se utilizan en el clob del procedimiento almacenado.

Tabla 63. Soporte para métodos javax.sql.CommonDataSource

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
getLoginTimeout	Sí	Sí
getLogWriter	Sí	Sí
getParentLogger1	Sí	No
setLoginTimeout	Sí	Sí
setLogWriter	Sí	Sí

Notas:

1. Esto es un método de JDBC 4.1.

Tabla 64. Soporte para métodos java.sql.Connection

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
abort[1]	Sí	No
clearWarnings	Sí	Sí
close	Sí	Sí
commit	Sí	Sí
createArrayOf[2]	Sí	No
createBlob[2]	Sí	No
createClob[2]	Sí	No
createStatement	Sí	Sí
createStruct[2]	Sí	No
getAutoCommit	Sí	Sí

Tabla 64. Soporte para métodos java.sql.Connection (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
getCatalog	Sí	Sí
getClientInfo[2]	Sí	No
getHoldability	Sí	No
getMetaData	Sí	Sí
getNetworkTimeout[1]	Sí	No
getSchema[1]	Sí	No
getTransactionIsolation	Sí	Sí
getTypeMap	No	Sí
getWarnings	Sí	Sí
isClosed	Sí	Sí
isReadOnly	Sí	Sí
isValid[2,3]	Sí	No
nativeSQL	Sí	Sí
prepareCall	Sí[4]	Sí
prepareStatement	Sí	Sí
releaseSavepoint	Sí	No
rollback	Sí	Sí
setAutoCommit	Sí	Sí
setCatalog	Sí	No
setClientInfo[2]	Sí	No
setNetworkTimeout[1]	Sí	No
setReadOnly	Sí[5]	No
setSavepoint	Sí	No
setSchema[1]	Sí	No
setTransactionIsolation	Sí	Sí
setTypeMap	No	Sí

Notas:

1. Esto es un método de JDBC 4.1.
2. Esto es un método de JDBC 4.0.
3. Cuando se utiliza IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4, se emite una excepción de SQL (SQLException) si el valor del parámetro *timeout* es menor que 0. Cuando se utiliza IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2, se emite una excepción de SQL (SQLException) si el valor del parámetro *timeout* es distinto de 0.
4. Si el procedimiento almacenado contenido en la sentencia CALL se ejecuta en DB2 para z/OS, los parámetros de la sentencia CALL no pueden ser expresiones.
5. El controlador no utiliza el valor. Para IBM Data Server Driver para JDBC y SQLJ, una conexión se puede definir como de sólo lectura mediante la propiedad `readOnly` de un objeto Connection o DataSource.

Tabla 65. Soporte para métodos javax.sql.ConnectionEvent

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
Métodos heredados de java.util.EventObject	Sí	Sí

Tabla 65. Soporte para métodos javax.sql.ConnectionEvent (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
getSQLException	Sí	Sí

Tabla 66. Soporte para métodos javax.sql.ConnectionEventListener

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
connectionClosed	Sí	Sí
connectionErrorOccurred	Sí	Sí

Tabla 67. Soporte para métodos javax.sql.ConnectionPoolDataSource

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
getLoginTimeout	Sí	Sí
getLogWriter	Sí	Sí
getPooledConnection	Sí	Sí
setLoginTimeout	Sí[1]	Sí
setLogWriter	Sí	Sí

Nota:

1. Este método no está soportado para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS.

Tabla 68. Soporte para métodos java.sql.DatabaseMetaData

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
allProceduresAreCallable	Sí	Sí
allTablesAreSelectable	Sí[1]	Sí[1]
dataDefinitionCausesTransactionCommit	Sí	Sí
dataDefinitionIgnoredInTransactions	Sí	Sí
deletesAreDetected	Sí	Sí
doesMaxRowSizeIncludeBlobs	Sí	Sí
generatedKeyAlwaysReturned[2]	Sí	No
getAttributes	Sí[3]	No
getBestRowIdentifier	Sí	Sí
getCatalogs	Sí	Sí
getCatalogSeparator	Sí	Sí
getCatalogTerm	Sí	Sí
getClientInfoProperties[7]	Sí	No
getColumnPrivileges	Sí	Sí
getColumns	Sí[8]	Sí[11]
getConnection	Sí	Sí
getCrossReference	Sí	Sí

Tabla 68. Soporte para métodos java.sql.DatabaseMetaData (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
getDatabaseMajorVersion	Sí	No
getDatabaseMinorVersion	Sí	No
getDatabaseProductName	Sí	Sí
getDatabaseProductVersion	Sí	Sí
getDefaultTransactionIsolation	Sí	Sí
getDriverMajorVersion	Sí	Sí
getDriverMinorVersion	Sí	Sí
getDriverName	Sí[9]	Sí
getDriverVersion	Sí	Sí
getExportedKeys	Sí	Sí
getFunctionColumns[7]	Sí	No
getFunctions[7]	Sí	No
getExtraNameCharacters	Sí	Sí
getIdentifierQuoteString	Sí	Sí
getImportedKeys	Sí	Sí
getIndexInfo	Sí	Sí
getJDBCMajorVersion	Sí	No
getJDBCMinorVersion	Sí	No
getMaxBinaryLiteralLength	Sí	Sí
getMaxCatalogNameLength	Sí	Sí
getMaxCharLiteralLength	Sí	Sí
getMaxColumnNameLength	Sí	Sí
getMaxColumnsInGroupBy	Sí	Sí
getMaxColumnsInIndex	Sí	Sí
getMaxColumnsInOrderBy	Sí	Sí
getMaxColumnsInSelect	Sí	Sí
getMaxColumnsInTable	Sí	Sí
getMaxConnections	Sí	Sí
getMaxCursorNameLength	Sí	Sí
getMaxIndexLength	Sí	Sí
getMaxProcedureNameLength	Sí	Sí
getMaxRowSize	Sí	Sí
getMaxSchemaNameLength	Sí	Sí
getMaxStatementLength	Sí	Sí
getMaxStatements	Sí	Sí
getMaxTableNameLength	Sí	Sí
getMaxTablesInSelect	Sí	Sí
getMaxUserNameLength	Sí	Sí

Tabla 68. Soporte para métodos java.sql.DatabaseMetaData (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
getNumericFunctions	Sí	Sí
getPrimaryKeys	Sí	Sí
getProcedureColumns	Sí8 en la página 438	Sí
getProcedures	Sí8 en la página 438	Sí
getProcedureTerm	Sí	Sí
getPseudoColumns[2]	Sí	No
getResultSetHoldability	Sí	No
getRowIdLifetime[7]	Sí	No
getSchemas	Sí10 en la página 438	Sí[11]
getSchemaTerm	Sí	Sí
getSearchStringEscape	Sí	Sí
getSQLKeywords	Sí	Sí
getSQLStateType	Sí	No
getStringFunctions	Sí	Sí
getSuperTables	Sí[3]	No
getSuperTypes	Sí[3]	No
getSystemFunctions	Sí	Sí
getTablePrivileges	Sí	Sí
getTables	Sí	Sí[11]
getTableTypes	Sí	Sí
getTimeDateFunctions	Sí	Sí
getTypeInfo	Sí	Sí
getUDTs	No	Sí[12]
getURL	Sí	Sí
getUserName	Sí	Sí
getVersionColumns	Sí	Sí
insertsAreDetected	Sí	Sí
isCatalogAtStart	Sí	Sí
isReadOnly	Sí	Sí
locatorsUpdateCopy	Sí[4]	Sí[4]
nullPlusNonNullIsNull	Sí	Sí
nullsAreSortedAtEnd	Sí[5]	Sí[5]
nullsAreSortedAtStart	Sí	Sí
nullsAreSortedHigh	Sí[6]	Sí[6]
nullsAreSortedLow	Sí[1]	Sí[1]
othersDeletesAreVisible	Sí	Sí

Tabla 68. Soporte para métodos java.sql.DatabaseMetaData (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
othersInsertsAreVisible	Sí	Sí
othersUpdatesAreVisible	Sí	Sí
ownDeletesAreVisible	Sí	Sí
ownInsertsAreVisible	Sí	Sí
ownUpdatesAreVisible	Sí	Sí
storesLowerCaseIdentifiers	Sí[1]	Sí[1]
storesLowerCaseQuotedIdentifiers	Sí[5]	Sí[5]
storesMixedCaseIdentifiers	Sí	Sí
storesMixedCaseQuotedIdentifiers	Sí	Sí
storesUpperCaseIdentifiers	Sí[6]	Sí[6]
storesUpperCaseQuotedIdentifiers	Sí	Sí
supportsAlterTableWithAddColumn	Sí	Sí
supportsAlterTableWithDropColumn	Sí[1]	Sí[1]
supportsANSI92EntryLevelSQL	Sí	Sí
supportsANSI92FullSQL	Sí	Sí
supportsANSI92IntermediateSQL	Sí	Sí
supportsBatchUpdates	Sí	Sí
supportsCatalogsInDataManipulation	Sí[1]	Sí[1]
supportsCatalogsInIndexDefinitions	Sí	Sí
supportsCatalogsInPrivilegeDefinitions	Sí	Sí
supportsCatalogsInProcedureCalls	Sí[1]	Sí[1]
supportsCatalogsInTableDefinitions	Sí	Sí
SupportsColumnAliasing	Sí	Sí
supportsConvert	Sí	Sí
supportsCoreSQLGrammar	Sí	Sí
supportsCorrelatedSubqueries	Sí	Sí
supportsDataDefinitionAndDataManipulationTransactions	Sí	Sí
supportsDataManipulationTransactionsOnly	Sí	Sí
supportsDifferentTableCorrelationNames	Sí[5]	Sí[5]
supportsExpressionsInOrderBy	Sí	Sí
supportsExtendedSQLGrammar	Sí	Sí
supportsFullOuterJoins	Sí[4]	Sí[4]
supportsGetGeneratedKeys	Sí	No
supportsGroupBy	Sí	Sí
supportsGroupByBeyondSelect	Sí	Sí
supportsGroupByUnrelated	Sí	Sí
supportsIntegrityEnhancementFacility	Sí	Sí
supportsLikeEscapeClause	Sí	Sí

Tabla 68. Soporte para métodos java.sql.DatabaseMetaData (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
supportsLimitedOuterJoins	Sí	Sí
supportsMinimumSQLGrammar	Sí	Sí
supportsMixedCaseIdentifiers	Sí	Sí
supportsMixedCaseQuotedIdentifiers	Sí[4]	Sí[4]
supportsMultipleOpenResults	Sí[6]	Sí[6]
supportsMultipleResultSets	Sí[6]	Sí[6]
supportsMultipleTransactions	Sí	Sí
supportsNamedParameters	Sí	No
supportsNonNullableColumns	Sí	Sí
supportsOpenCursorsAcrossCommit	Sí[4]	Sí[4]
supportsOpenCursorsAcrossRollback	Sí	Sí
supportsOpenStatementsAcrossCommit	Sí[4]	Sí[4]
supportsOpenStatementsAcrossRollback	Sí[4]	Sí[4]
supportsOrderByUnrelated	Sí	Sí
supportsOuterJoins	Sí	Sí
supportsPositionedDelete	Sí	Sí
supportsPositionedUpdate	Sí	Sí
supportsResultSetConcurrency	Sí	Sí
supportsResultSetHoldability	Sí	No
supportsResultSetType	Sí	Sí
supportsSavepoints	Sí	Sí
supportsSchemasInDataManipulation	Sí	Sí
supportsSchemasInIndexDefinitions	Sí	Sí
supportsSchemasInPrivilegeDefinitions	Sí	Sí
supportsSchemasInProcedureCalls	Sí	Sí
supportsSchemasInTableDefinitions	Sí	Sí
supportsSelectForUpdate	Sí	Sí
supportsStoredProcedures	Sí	Sí
supportsSubqueriesInComparisons	Sí	Sí
supportsSubqueriesInExists	Sí	Sí
supportsSubqueriesInIns	Sí	Sí
supportsSubqueriesInQuantifieds	Sí	Sí
supportsSuperTables	Sí	No
supportsSuperTypes	Sí	No
supportsTableCorrelationNames	Sí	Sí
supportsTransactionIsolationLevel	Sí	Sí
supportsTransactions	Sí	Sí
supportsUnion	Sí	Sí

Tabla 68. Soporte para métodos java.sql.DatabaseMetaData (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
supportsUnionAll	Sí	Sí
updatesAreDetected	Sí	Sí
usesLocalFilePerTable	Sí	Sí
usesLocalFiles	Sí	Sí

Notas:

1. Las fuentes de datos DB2 devuelven `false` para este método. Las fuentes de datos IBM Informix devuelven `true`.
2. Esto es un método de JDBC 4.1.
3. Este método solamente se puede utilizar para conexiones con DB2 Database para Linux, UNIX y Windows e IBM Informix.
4. Cuando se utiliza IBM Data Server Driver para JDBC y SQLJ, las fuentes de datos DB2 y las fuentes de datos IBM Informix devuelven `true` para este método. Cuando se utiliza el controlador JDBC de IBM Informix, las fuentes de datos IBM Informix devuelven `false`.
5. Cuando se utiliza IBM Data Server Driver para JDBC y SQLJ, las fuentes de datos DB2 y las fuentes de datos IBM Informix devuelven `false` para este método. Cuando se utiliza el controlador JDBC de IBM Informix, las fuentes de datos IBM Informix devuelven `true`.
6. Las fuentes de datos DB2 devuelven `true` para este método. Las fuentes de datos IBM Informix devuelven `false`.
7. Esto es un método de JDBC 4.0.
8. Este método devuelve la columna adicional descrita por la especificación JDBC 4.0.
9. JDBC 3.0 e implementaciones anteriores de IBM Data Server Driver para JDBC y SQLJ devuelven "IBM DB2 JDBC Universal Driver Architecture."

   La implementación para JDBC 4.0 de IBM Data Server Driver para JDBC y SQLJ devuelve "IBM Data Server Driver para JDBC y SQLJ."
10. Se pueden utilizar el formato JDBC 4.0 y formatos anteriores de este método.
11. El controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows no es compatible con el formato JDBC 3.0 de este método.
12. El método se puede ejecutar, pero devuelve un ResultSet vacío.

Tabla 69. Soporte para métodos java.sql.DataSource

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
getConnection	Sí	Sí
getLoginTimeout	Sí	Sí
getLogWriter	Sí	Sí
setLoginTimeout	Sí[1]	Sí
setLogWriter	Sí	Sí

Notas:

1. Este método no está soportado para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS.

Tabla 70. Soporte para métodos java.sql.DataTruncation

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
Métodos heredados de java.lang.Throwable	Sí	Sí
Métodos heredados de java.sql.SQLException	Sí	Sí
Métodos heredados de java.sql.SQLWarning	Sí	Sí
getDataSize	Sí	Sí
getIndex	Sí	Sí
getParameter	Sí	Sí
getRead	Sí	Sí
getTransferSize	Sí	Sí

Tabla 71. Soporte para métodos java.sql.Driver

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
acceptsURL	Sí	Sí
connect	Sí	Sí
getMajorVersion	Sí	Sí
getMinorVersion	Sí	Sí
getParentLogger	Sí	No
getPropertyInfo	Sí	Sí
jdbcCompliant	Sí	Sí

Tabla 72. Soporte para métodos java.sql.DriverManager

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
deregisterDriver	Sí	Sí
getConnection	Sí	Sí
getDriver	Sí	Sí
getDrivers	Sí	Sí
getLoginTimeout	Sí	Sí
getLogStream	Sí	Sí
getLogWriter	Sí	Sí
println	Sí	Sí
registerDriver	Sí	Sí
setLoginTimeout	Sí[1]	Sí
setLogStream	Sí	Sí
setLogWriter	Sí	Sí

Notas:

1. Este método no está soportado para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS.

Tabla 73. Soporte para métodos java.sql.ParameterMetaData

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
getParameterClassName	No	No
getParameterCount	Sí	No
getParameterMode	Sí	No
getParameterType	Sí	No
getParameterTypeName	Sí	No
getPrecision	Sí	No
getScale	Sí	No
isNullable	Sí	No
isSigned	Sí	No

Tabla 74. Soporte para métodos javax.sql.PooledConnection

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
addConnectionEventListener	Sí	Sí
addStatementEventListener[1]	Sí	No
close	Sí	Sí
getConnection	Sí	Sí
removeConnectionEventListener	Sí	Sí
removeStatementEventListener[1]	Sí	No

Notas:

1. Esto es un método de JDBC 4.0.

Tabla 75. Soporte para métodos java.sql.PreparedStatement

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
Métodos heredados de java.sql.Statement	Sí	Sí
addBatch	Sí	Sí
clearParameters	Sí	Sí
execute	Sí	Sí
executeQuery	Sí	Sí
executeUpdate	Sí	Sí
getMetaData	Sí	Sí
getParameterMetaData	Sí	Sí
setArray	No	No
setAsciiStream	Sí[1,2]	Sí
setBigDecimal	Sí	Sí
setBinaryStream	Sí[1,3]	Sí
setBlob	Sí[4]	Sí
setBoolean	Sí	Sí
setByte	Sí	Sí

Tabla 75. Soporte para métodos java.sql.PreparedStatement (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
setBytes	Sí	Sí
setCharacterStream	Sí[1,5]	Sí
setClob	Sí[6]	Sí
setDate	Sí[8]	Sí[8]
setDouble	Sí	Sí
setFloat	Sí	Sí
setInt	Sí	Sí
setLong	Sí	Sí
setNull	Sí[9]	Sí[9]
setObject	Sí	Sí
setRef	No	No
setRowId[7]	Sí	No
setShort	Sí	Sí
setString	Sí[10]	Sí[10]
setTime	Sí[8]	Sí[8]
setTimestamp	Sí[8]	Sí[8]
setUnicodeStream	Sí	Sí
setURL	Sí	Sí

Notas:

1. Si el valor del parámetro *length* es -1, se leen todos los datos de InputStream o Reader y se envían a la fuente de datos.
2. Los formatos válidos de este método incluyen los formatos siguientes de JDBC 4.0:
   ```
   setAsciiStream(int índiceParámetro, InputStream x, long longitud)
   setAsciiStream(int índiceParámetro, InputStream x)
   ```
3. Los formatos válidos de este método incluyen los formatos siguientes de JDBC 4.0:
   ```
   setBinaryStream(int índiceParámetro, InputStream x, long longitud)
   setBinaryStream(int índiceParámetro, InputStream x)
   ```
4. Los formatos válidos de este método incluyen el siguiente formato de JDBC 4.0:
   ```
   setBlob(int índiceParámetro, InputStream corrienteEntrada, long longitud)
   ```
5. Los formatos válidos de este método incluyen los formatos siguientes de JDBC 4.0:
   ```
   setCharacterStream(int índiceParámetro, Reader lector, long longitud)
   setCharacterStream(int índiceParámetro, Reader lector)
   ```
6. Los formatos válidos de este método incluyen el siguiente formato de JDBC 4.0:
   ```
   setClob(int índiceParámetro, Reader lector, long longitud)
   ```
7. Esto es un método de JDBC 4.0.
8. El servidor de bases de datos no realiza ajustes de zona horaria para valores de fecha y hora. El controlador JDBC ajusta un valor para la zona horaria local antes de enviar el valor al servidor si el usuario especifica un formato del método setDate, setTime o setTimestamp que incluye un parámetro java.util.Calendar.
9. El formato siguiente de setNull no se soporta:
   ```
   setNull(int Índiceparámetros, int TipoJdbc, String nombreTipo)
   ```
10. setString no está soportado si la columna tiene el atributo FOR BIT DATA o el tipo de datos es BLOB.

Tabla 76. Soporte para métodos java.sql.Ref

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
get BaseTypeName	No	No

Tabla 77. Soporte para métodos java.sql.ResultSet

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
absolute	Sí	Sí
afterLast	Sí	Sí
beforeFirst	Sí	Sí
cancelRowUpdates	Sí	No
clearWarnings	Sí	Sí
close	Sí	Sí
deleteRow	Sí	No
findColumn	Sí	Sí
first	Sí	Sí
getArray	No	No
getAsciiStream	Sí	Sí
getBigDecimal	Sí	Sí
getBinaryStream	Sí[1]	Sí
getBlob	Sí	Sí
getBoolean	Sí	Sí
getByte	Sí	Sí
getBytes	Sí	Sí
getCharacterStream	Sí	Sí
getClob	Sí	Sí
getConcurrency	Sí	Sí
getCursorName	Sí	Sí
getDate	Sí[3]	Sí[3]
getDouble	Sí	Sí
getFetchDirection	Sí	Sí
getFetchSize	Sí	Sí
getFloat	Sí	Sí
getInt	Sí	Sí
getLong	Sí	Sí
getMetaData	Sí	Sí
getObject	Sí[4]	Sí[4]
getRef	No	No
getRow	Sí	Sí
getRowId[10]	Sí	No
getShort	Sí	Sí
getStatement	Sí	Sí

Tabla 77. Soporte para métodos java.sql.ResultSet (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
getString	Sí	Sí
getTime	Sí[3]	Sí[3]
getTimestamp	Sí[3]	Sí[3]
getType	Sí	Sí
getUnicodeStream	Sí	Sí
getURL	Sí	Sí
getWarnings	Sí	Sí
insertRow	Sí	No
isAfterLast	Sí	Sí
isBeforeFirst	Sí	Sí
isFirst	Sí	Sí
isLast	Sí	Sí
last	Sí	Sí
moveToCurrentRow	Sí	No
moveToInsertRow	Sí	No
next	Sí	Sí
previous	Sí	Sí
refreshRow	Sí	No
relative	Sí	Sí
rowDeleted	Sí	No
rowInserted	Sí	No
rowUpdated	Sí	No
setFetchDirection	Sí	Sí
setFetchSize	Sí	Sí
updateArray	No	No
updateAsciiStream	Sí[5]	No
updateBigDecimal	Sí	No
updateBinaryStream	Sí[6]	No
updateBlob	Sí[7]	No
updateBoolean	Sí	No
updateByte	Sí	No
updateBytes	Sí	No
updateCharacterStream	Sí[8]	No
updateClob	Sí[9]	No
updateDate	Sí	No
updateDouble	Sí	No
updateFloat	Sí	No
updateInt	Sí	No
updateLong	Sí	No

Tabla 77. Soporte para métodos java.sql.ResultSet (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
updateNull	Sí	No
updateObject	Sí	No
updateRef	No	No
updateRow	Sí	No
updateRowId[10]	Sí	No
updateShort	Sí	No
updateString	Sí	No
updateTime	Sí	No
updateTimestamp	Sí	No
wasNull	Sí	Sí

Notas:

1. No se da soporte agetBinaryStream en columnas de tipo CLOB.
2. getMetaData rellena el nombre del esquema, si el nombre del esquema devuelto es inferior a 8 caracteres, para llegar a 8 caracteres.
3. El servidor de bases de datos no realiza ajustes de zona horaria para valores de fecha y hora. El controlador JDBC ajusta un valor para la zona horaria local después de recuperar el valor a partir del servidor si el usuario especifica un formato del método getDate, getTime o getTimestamp que incluye un parámetro java.util.Calendar.
4. El formato siguiente del método getObject no se soporta:
   ```
   getObject(int Índiceparámetros, java.util.Map
   map)
   ```
5. Los formatos válidos de este método incluyen los formatos siguientes de JDBC 4.0:
   ```
   updateAsciiStream(int índiceColumna, InputStream x)
   updateAsciiStream(String etiquetaColumna, InputStream x)
   updateAsciiStream(int índiceColumna, InputStream x, long longitud)
   updateAsciiStream(String etiquetaColumna, InputStream x, long longitud)
   ```
6. Los formatos válidos de este método incluyen los formatos siguientes de JDBC 4.0:
   ```
   updateBinaryStream(int índiceColumna, InputStream x)
   updateBinaryStream(String etiquetaColumna, InputStream x)
   updateBinaryStream(int índiceColumna, InputStream x, long longitud)
   updateBinaryStream(String etiquetaColumna, InputStream x, long longitud)
   ```
7. Los formatos válidos de este método incluyen los formatos siguientes de JDBC 4.0:
   ```
   updateBlob(int índiceColumna, InputStream x)
   updateBlob(String etiquetaColumna, InputStream x)
   updateBlob(int índiceColumna, InputStream x, long longitud)
   updateBlob(String etiquetaColumna, InputStream x, long longitud)
   ```
8. Los formatos válidos de este método incluyen los formatos siguientes de JDBC 4.0:
   ```
   updateCharacterStream(int índiceColumna, Reader lector)
   updateCharacterStream(String etiquetaColumna, Reader lector)
   updateCharacterStream(int índiceColumna, Reader lector, long longitud)
   updateCharacterStream(String etiquetaColumna, Reader lector, long longitud)
   ```
9. Los formatos válidos de este método incluyen los formatos siguientes de JDBC 4.0:
   ```
   updateClob(int índiceColumna, Reader lector)
   updateClob(String etiquetaColumna, Reader lector)
   updateClob(int índiceColumna, Reader lector, long longitud)
   updateClob(String etiquetaColumna, Reader lector, long longitud)
   ```
10. Esto es un método de JDBC 4.0.

Tabla 78. Soporte para métodos java.sql.ResultSetMetaData

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
getCatalogName	Sí	Sí
getColumnClassName	No	Sí
getColumnCount	Sí	Sí
getColumnDisplaySize	Sí	Sí
getColumnLabel	Sí	Sí
getColumnName	Sí	Sí
getColumnType	Sí	Sí
getColumnTypeName	Sí	Sí
getPrecision	Sí	Sí
getScale	Sí	Sí
getSchemaName	Sí	Sí
getTableName	Sí[1]	Sí
isAutoIncrement	Sí	Sí
isCaseSensitive	Sí	Sí
isCurrency	Sí	Sí
isDefinitelyWritable	Sí	Sí
isNullable	Sí	Sí
isReadOnly	Sí	Sí
isSearchable	Sí	Sí
isSigned	Sí	Sí
isWritable	Sí	Sí

Notas:

1. Para las fuentes de datos IBM Informix, getTableName no devuelve un valor.
2. getSchemaName rellena el nombre del esquema, si el nombre del esquema devuelto es inferior a 8 caracteres, para llegar a 8 caracteres.

Tabla 79. Soporte para métodos java.sql.RowId[1]

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ[2]	Soporte del controlador JDBC de IBM Informix
equals	Sí	No
getBytes	Sí	No
hashCode	No	No
toString	Sí	No

Notas:

1. Estos métodos son métodos de JDBC 4.0.
2. Estos métodos reciben soporte para las conexiones con fuentes de datos DB2 para z/OS, DB2 para i e IBM Informix.

Tabla 80. Soporte para métodos java.sql.SQLClientInfoException[1]

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
Métodos heredados de java.lang.Exception	Sí	No
Métodos heredados de java.lang.Throwable	Sí	No
Métodos heredados de java.lang.Object	Sí	No
getFailedProperties	Sí	No

Nota:

1. Esto es una clase de JDBC 4.0.

Tabla 81. Soporte para métodos java.sql.SQLData

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
getSQLTypeName	No	No
readSQL	No	No
writeSQL	No	No

Tabla 82. Soporte para métodos java.sql.SQLDataException[1]

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
Métodos heredados de java.lang.Exception	Sí	No
Métodos heredados de java.lang.Throwable	Sí	No
Métodos heredados de java.lang.Object	Sí	No

Nota:

1. Esto es una clase de JDBC 4.0.

Tabla 83. Soporte para métodos java.sql.SQLDataException[1]

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
Métodos heredados de java.lang.Exception	Sí	No
Métodos heredados de java.lang.Throwable	Sí	No
Métodos heredados de java.lang.Object	Sí	No

Nota:

1. Esto es una clase de JDBC 4.0.

Tabla 84. Soporte para métodos java.sql.SQLException

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
Métodos heredados de java.lang.Exception	Sí	Sí
getSQLState	Sí	Sí
getErrorCode	Sí	Sí
getNextException	Sí	Sí
setNextException	Sí	Sí

Tabla 85. Soporte para métodos java.sql.SQLFeatureNotSupported[1]

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
Métodos heredados de java.lang.Exception	Sí	No
Métodos heredados de java.lang.Throwable	Sí	No
Métodos heredados de java.lang.Object	Sí	No

Nota:

1. Esto es una clase de JDBC 4.0.

Tabla 86. Soporte para métodos java.sql.SQLInput

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
readArray	No	No
readAsciiStream	No	No
readBigDecimal	No	No
readBinaryStream	No	No
readBlob	No	No
readBoolean	No	No
readByte	No	No
readBytes	No	No
readCharacterStream	No	No
readClob	No	No
readDate	No	No
readDouble	No	No
readFloat	No	No
readInt	No	No
readLong	No	No
readObject	No	No
readRef	No	No
readShort	No	No
readString	No	No
readTime	No	No
readTimestamp	No	No
wasNull	No	No

Tabla 87. Soporte para métodos java.sql.SQLIntegrityConstraintViolationException[1]

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
Métodos heredados de java.lang.Exception	Sí	No
Métodos heredados de java.lang.Throwable	Sí	No
Métodos heredados de java.lang.Object	Sí	No

Nota:

1. Esto es una clase de JDBC 4.0.

Tabla 88. Soporte para métodos java.sql.SQLInvalidAuthorizationSpecException[1]

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
Métodos heredados de java.lang.Exception	Sí	No
Métodos heredados de java.lang.Throwable	Sí	No
Métodos heredados de java.lang.Object	Sí	No

Nota:

1. Esto es una clase de JDBC 4.0.

Tabla 89. Soporte para métodos java.sql.SQLNonTransientConnectionException[1]

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
Métodos heredados de java.lang.Exception	Sí	No
Métodos heredados de java.lang.Throwable	Sí	No
Métodos heredados de java.lang.Object	Sí	No

Nota:

1. Esto es una clase de JDBC 4.0.

Tabla 90. Soporte para métodos java.sql.SQLNonTransientException[1]

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
Métodos heredados de java.lang.Exception	Sí	No
Métodos heredados de java.lang.Throwable	Sí	No
Métodos heredados de java.lang.Object	Sí	No

Nota:

1. Esto es una clase de JDBC 4.0.

Tabla 91. Soporte para métodos java.sql.SQLOutput

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
writeArray	No	No
writeAsciiStream	No	No
writeBigDecimal	No	No
writeBinaryStream	No	No
writeBlob	No	No
writeBoolean	No	No
writeByte	No	No
writeBytes	No	No
writeCharacterStream	No	No
writeClob	No	No
writeDate	No	No
writeDouble	No	No
writeFloat	No	No
writeInt	No	No

Tabla 91. Soporte para métodos java.sql.SQLOutput (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
writeLong	No	No
writeObject	No	No
writeRef	No	No
writeShort	No	No
writeString	No	No
writeStruct	No	No
writeTime	No	No
writeTimestamp	No	No

Tabla 92. Soporte para métodos java.sql.SQLRecoverableException[1]

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
Métodos heredados de java.lang.Exception	Sí	No
Métodos heredados de java.lang.Throwable	Sí	No
Métodos heredados de java.lang.Object	Sí	No

Nota:

1. Esto es una clase de JDBC 4.0.

Tabla 93. Soporte para métodos java.sql.SQLSyntaxErrorException[1]

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
Métodos heredados de java.lang.Exception	Sí	No
Métodos heredados de java.lang.Throwable	Sí	No
Métodos heredados de java.lang.Object	Sí	No

Nota:

1. Esto es una clase de JDBC 4.0.

Tabla 94. Soporte para métodos java.sql.SQLTimeoutException[1]

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
Métodos heredados de java.lang.Exception	Sí	No
Métodos heredados de java.lang.Throwable	Sí	No
Métodos heredados de java.lang.Object	Sí	No

Nota:

1. Esto es una clase de JDBC 4.0.

Tabla 95. Soporte para métodos java.sql.SQLTransientConnectionException[1]

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
Métodos heredados de java.lang.Exception	Sí	No
Métodos heredados de java.lang.Throwable	Sí	No

Tabla 95. Soporte para métodos java.sql.SQLTransientConnectionException[1] (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
Métodos heredados de java.lang.Object	Sí	No

Nota:

1. Esto es una clase de JDBC 4.0.

Tabla 96. Soporte para métodos java.sql.SQLTransientException[1]

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
Métodos heredados de java.lang.Exception	Sí	No
Métodos heredados de java.lang.Throwable	Sí	No
Métodos heredados de java.lang.Object	Sí	No

Nota:

1. Esto es una clase de JDBC 4.0.

Tabla 97. Soporte para métodos java.sql.SQLTransientRollbackException[1]

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
Métodos heredados de java.lang.Exception	Sí	No
Métodos heredados de java.lang.Throwable	Sí	No
Métodos heredados de java.lang.Object	Sí	No

Nota:

1. Esto es una clase de JDBC 4.0.

Tabla 98. Soporte para métodos java.sql.SQLXML[1]

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
free	Sí	No
getBinaryStream	Sí	No
getCharacterStream	Sí	No
getSource	Sí	No
getString	Sí	No
setBinaryStream	Sí	No
setCharacterStream	Sí	No
setResult	Sí	No
setString	Sí	No

Notas:

1. Estos métodos son métodos de JDBC 4.0. Estos métodos no se pueden utilizar para conexiones con servidores IBM Informix.

Tabla 99. Soporte para métodos java.sql.Statement

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
abort[1]	Sí	No

Tabla 99. Soporte para métodos java.sql.Statement (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
addBatch	Sí	Sí
cancel	Sí[2]	Sí
clearBatch	Sí	Sí
clearWarnings	Sí	Sí
close	Sí	Sí
closeOnCompletion[1]	Sí	No
execute	Sí	Sí
executeBatch	Sí	Sí
executeQuery	Sí	Sí
executeUpdate	Sí	Sí
getConnection	Sí	Sí
getFetchDirection	Sí	Sí
getFetchSize	Sí	Sí
getGeneratedKeys	Sí	No
getMaxFieldSize	Sí	Sí
getMaxRows	Sí	Sí
getMoreResults	Sí	Sí
getQueryTimeout	Sí[6,5]	Sí
getResultSet	Sí	Sí
getResultSetConcurrency	Sí	Sí
getResultSetHoldability	Sí	No
getResultSetType	Sí	Sí
getUpdateCount[3]	Sí	Sí
getWarnings	Sí	Sí
isCloseOnCompletion[1]	Sí	No
isClosed[7]	Sí	No
isPoolable[7]	Sí	No
setCursorName	Sí	Sí
setEscapeProcessing	Sí	Sí
setFetchDirection	Sí	Sí
setFetchSize	Sí	Sí
setMaxFieldSize	Sí	Sí
setMaxRows	Sí	Sí
setPoolable[7]	Sí	No
setQueryTimeout	Sí[4,6,5]	Sí

Tabla 99. Soporte para métodos java.sql.Statement (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix

Notas:

1. Esto es un método de JDBC 4.1.
2. Con IBM Data Server Driver para JDBC y SQLJ, únicamente se da soporte a Statement.cancel en los siguientes entornos:
   - Conectividad de tipo 2 y tipo 4 desde un cliente Linux, UNIX o Windows a un servidor DB2 Database para Linux, UNIX y Windows versión 8 o posterior.
   - Conectividad de tipo 2 y tipo 4 desde un cliente Linux, UNIX o Windows a un servidor DB2 para z/OS versión 9 o posterior.
   - Conectividad de tipo 4 desde un cliente z/OS a un servidor DB2 Database para Linux, UNIX y Windows versión 8 o posterior.
   - Conectividad de tipo 4 desde un cliente z/OS a un servidor DB2 para z/OS versión 8 o posterior.

   La acción que efectúa IBM Data Server Driver para JDBC y SQLJ cuando la aplicación ejecuta Statement.cancel también depende del valor de la propiedad DB2BaseDataSource.interruptProcessingMode.
3. No está soportado para ResultSet de procedimiento almacenado.
4. En el caso de DB2 para i, este método se puede utilizar solamente para un valor de *segundos* igual a 0.
5. Para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS, solamente se da soporte a Statement.setQueryTimeout si la propiedad queryTimeoutInterruptProcessingMode de Connection o DataSource está establecida en INTERRUPT_PROCESSING_MODE_CLOSE_SOCKET.
6. Para IBM Data Server Driver para JDBC y SQLJ versión 4.0 y posterior, se admite Statement.setQueryTimeout para los siguientes métodos:
   - Statement.execute
   - Statement.executeUpdate
   - Statement.executeQuery

   Statement.setQueryTimeout no se admite para el método Statement.executeBatch.
7. Esto es un método de JDBC 4.0.

Tabla 100. Soporte para métodos java.sql.Struct

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
getSQLTypeName	No	No
getAttributes	No	No

Tabla 101. Soporte para métodos java.sql.Wrapper

Método JDBC[1]	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
isWrapperFor	Sí	No
unWrap	Sí	No

Notas:

1. Estos métodos son métodos de JDBC 4.0.

Tabla 102. Soporte para métodos javax.sql.XAConnection

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ[1]	Soporte del controlador JDBC de IBM Informix
Métodos heredados de javax.sql.PooledConnection	Sí	Sí
getXAResource	Sí	Sí

Tabla 102. Soporte para métodos javax.sql.XAConnection (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ[1]	Soporte del controlador JDBC de IBM Informix

Notas:

1. Estos métodos se pueden utilizar para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 con un servidor DB2 Database para Linux, UNIX y Windows o IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4.

Tabla 103. Soporte para métodos javax.sql.XADataSource

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
getLoginTimeout	Sí	Sí
getLogWriter	Sí	Sí
getXAConnection	Sí	Sí
setLoginTimeout	Sí	Sí
setLogWriter	Sí	Sí

Tabla 104. Soporte para métodos javax.transaction.xa.XAResource

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de IBM Informix
commit	Sí[1]	Sí
end	Sí[1,2]	Sí
forget	Sí[1]	Sí
getTransactionTimeout	Sí[3]	Sí
isSameRM	Sí[1]	Sí
prepare	Sí[1]	Sí
recover	Sí[1]	Sí
rollback	Sí[1]	Sí
setTransactionTimeout	Sí[3]	Sí
start	Sí[1]	Sí

Notas:

1. Se da soporte a este método para la IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 con un servidor DB2 Database para Linux, UNIX y Windows o una IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4.
2. Cuando se llama al método end, IBM Data Server Driver para JDBC y SQLJ cierra el cursor subyacente, aunque esté especificado el distintivo TMSUSPEND.
3. Este método se puede utilizar para la IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 con DB2 Database para Linux, UNIX y Windows Versión 9.1 o posterior.

Soporte de IBM Data Server Driver para JDBC y SQLJ para la sintaxis de escape de SQL

IBM Data Server Driver para JDBC y SQLJ da soporte a la sintaxis de escape de SQL, como se describe en la especificación JDBC 1.0.

Se trata de la misma sintaxis que la que se utiliza en las cláusulas de escape del proveedor de las aplicaciones ODBC y CLI.

La sintaxis de escape de SQL está soportada en las aplicaciones JDBC y SQLJ.

Información de consulta sobre sentencias de SQLJ

Las sentencias de SQLJ se utilizan para el control de transacciones y la ejecución de sentencias de SQL.

Cláusula SQLJ

Las sentencias de SQL de un programa SQLJ están contenidas en cláusulas SQLJ.

Sintaxis

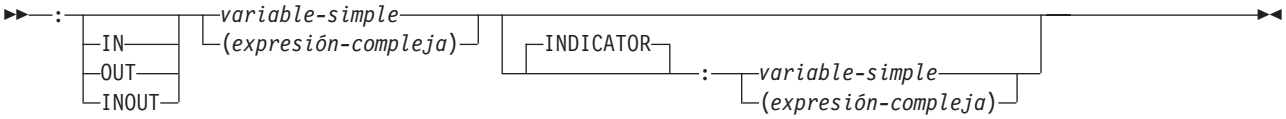
Notas de uso

Las palabras clave de una cláusula SQLJ distinguen entre mayúsculas y minúsculas, a menos que esas palabras clave formen parte de una sentencia de SQL dentro de una cláusula ejecutable.

Expresión de lenguaje principal de SQLJ

Una expresión de lenguaje principal es una variable o expresión Java que se especifica en cláusulas de SQLJ dentro de un programa de aplicación SQLJ.

Sintaxis



Descripción

: Indica que la variable o expresión que sigue a continuación es una expresión de lenguaje principal. La variable o expresión debe estar precedida inmediatamente por los dos puntos (:).

IN|OUT|INOUT
Para expresiones de lenguaje principal que se utilizan como parámetros en una llamada de procedimiento almacenado, identifica si el parámetro proporciona datos al procedimiento almacenado (IN), recibe datos del procedimiento almacenado (OUT), o realiza ambas cosas (INOUT). El valor por omisión es IN.

variable-simple
Especifica un identificador Java no calificado.

expresión-compleja
Especifica una expresión Java cuya evaluación da como resultado un valor individual.

INDICATOR :variable-simple o INDICATOR :(expresión-compleja)
Especifica la variable de indicador opcional para la variable de lenguaje principal Java correspondiente. El tipo de datos de la variable de indicador

debe ser el tipo SHORT de Java. Los únicos valores válidos para :variable-simple o :(expresión-compleja) son:

Para las aplicaciones personalizadas y para la entrada, solamente son válidos los valores siguientes:

Valor de indicador	Constante equivalente	Significado del valor
-1	sqlj.runtime.ExecutionContext.DBNull	Nulo
-2, -3, -4, -6		Nulo
-5	sqlj.runtime.ExecutionContext.DBDefault	Por omisión
-7	sqlj.runtime.ExecutionContext.DBUnassigned	Sin asignar
valor-short >=0	sqlj.runtime.ExecutionContext.DBNonNull	No nulo

Para las aplicaciones no personalizadas y para la entrada, solamente son válidos los valores siguientes:

Valor de indicador	Constante equivalente	Significado del valor
-1	sqlj.runtime.ExecutionContext.DBNull	Nulo
-7 <= *valor-short* < -1		Nulo
0	sqlj.runtime.ExecutionContext.DBNonNull	No nulo
valor-short >0		No nulo

Para las aplicaciones personalizadas o no personalizadas y para la salida, SQLJ establece uno de los valores siguientes:

Valor de indicador	Constante equivalente	Significado del valor
-1	sqlj.runtime.ExecutionContext.DBNull	El valor recuperado es SQL NULL
0		El valor recuperado no es SQL NULL

Notas de uso

- Las expresiones complejas deben estar encerradas entre paréntesis.
- La ubicación de una expresión de lenguaje principal dentro de una sentencia de SQL estático está regida por reglas de ANSI/ISO.
- Las variables de indicador son necesarias en los casos siguientes:
  - Para la entrada, cuando un tipo Java primitivo se utiliza para asignar el valor NULL a una columna de tabla.
  - Para la salida, cuando un tipo Java primitivo se utiliza para una variable de lenguaje principal y la columna de fuente puede devolver valores NULL.

    Si se devuelve un valor NULL de SQL y no se ha definido ninguna variable de indicador, se genera una excepción SQLNullException.

  Las variables de indicador no son necesarias para la entrada o salida de un valor nulo de Java como valor NULL de SQL, si el tipo de datos de la variable del lenguaje principal es:
  - El tipo de datos de una clase Java
  - Un tipo de base de datos personalizado admitido por el controlador
- , ... *variable-n*
- Para la salida, las variables de indicador son válidas en los siguientes tipos de sentencias:
  - Sentencia CALL con parámetros OUT o INOUT

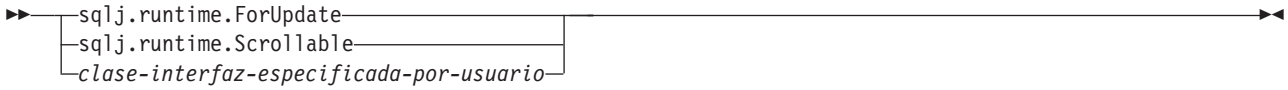
- FETCH *iterador-posición* INTO *variable-1, ... variable-n*
- SELECT *columna-1, ... columna-n* INTO *variable-1, ... variable-n*

Cláusula implements de SQLJ

La cláusula implements obtiene una o varias clases a partir de una interfaz Java.

Sintaxis

elemento-interfaz:



Descripción

elemento-interfaz
Especifica una interfaz Java definida por el usuario, la interfaz sqlj.runtime.ForUpdate de SQLJ o la interfaz sqlj.runtime.Scrollable de SQLJ.

Es necesario que implemente sqlj.runtime.ForUpdate cuando declare un iterador para una operación UPDATE o DELETE de posición. Consulte "Ejecutar operaciones UPDATE y DELETE de posición en una aplicación SQLJ" para obtener información sobre la ejecución de una operación UPDATE o DELETE de posición en SQLJ.

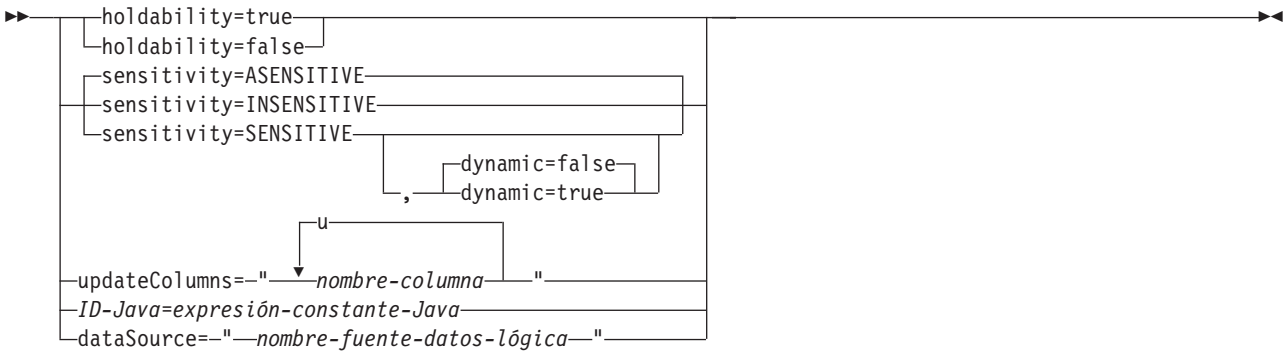
Es necesario que implemente sqlj.runtime.Scrollable cuando declare un iterador desplazable. Consulte "Utilizar iteradores desplazables en una aplicación SQLJ" para obtener información sobre iteradores desplazables.

Cláusula with de SQLJ

La cláusula with especifica uno o más atributos para un iterador o contexto de conexión.

Sintaxis

elemento-with:



Descripción

capacidad de retención
Para un iterador, especifica si el iterador conserva su posición en una tabla después de ejecutarse una operación COMMIT. El valor de holdability debe ser `true` o `false`.

sensitivity
Para un iterador, especifica si los cambios hechos en la tabla subyacente pueden ser visibles para el iterador después de abrir el iterador. El valor debe ser INSENSITIVE, SENSITIVE o ASENSITIVE. El valor por omisión es ASENSITIVE.

Para conexiones con IBM Informix, solamente se puede utilizar el valor INSENSITIVE.

dynamic
Para un iterador que esté definido con sensitivity=SENSITIVE, este atributo especifica si se cumplen las condiciones siguientes:

- Cuando la aplicación ejecuta sentencias UPDATE y DELETE de posición con el iterador, esos cambios son visibles para el iterador.
- Cuando la aplicación ejecuta sentencias INSERT, UPDATE y DELETE dentro de la aplicación, pero fuera del iterador, esos cambios son visibles para el iterador.

El valor de dynamic debe ser `true` o `false`. El valor por omisión es `false`.

Los servidores DB2 Database para Linux, UNIX y Windows no dan soporte a los cursores desplazables dinámicos. Especifique `true` sólo si la aplicación accede a datos en servidores DB2 para z/OS en la versión 9 o posterior.

Para conexiones con IBM Informix, solamente se puede utilizar el valor `false`. IBM Informix no es compatible con cursores dinámicos.

updateColumns
Para un iterador, especifica las columnas que se deben modificar cuando el iterador se utiliza para una sentencia UPDATE de posición. El valor de updateColumns debe ser una serie de caracteres literal que contenga los nombres de las columnas, separados por comas.

nombre-columna
Para un iterador, especifica una columna de la tabla de resultados que se debe actualizar utilizando el iterador.

ID-Java
Para un iterador o contexto de conexión, especifica una variable Java que

identifica un atributo definido por el usuario del iterador o contexto de conexión. El valor de *expresión-constante-Java* también está definido por el usuario.

dataSource
Para un contexto de conexión, especifica el nombre lógico de un objeto DataSource, creado por separado, que representa la fuente de datos a la que se conectará la aplicación. Esta opción solo está disponible para IBM Data Server Driver para JDBC y SQLJ.

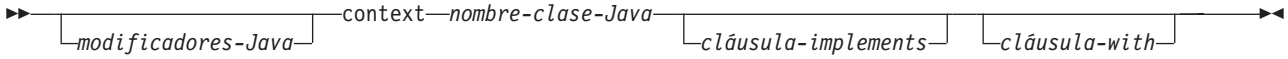
Notas de uso

- El valor situado en el lado izquierdo de un elemento with debe ser exclusivo dentro de su cláusula.
- Si especifica updateColumns en un elemento with de una cláusula de declaración de iterador, esta cláusula también debe contener una cláusula implements en la que se especifique la interfaz sqlj.runtime.ForUpdate.
- Si el usuario no personaliza su programa SQLJ, el controlador JDBC no tiene en cuenta el valor de holdability que está contenido en la cláusula with. En lugar de ello, el controlador utiliza el valor de holdability definido para el controlador JDBC.

Cláusula de declaración de conexión de SQLJ

La cláusula de declaración de conexión declara una conexión con una fuente de datos en un programa de aplicación de SQLJ.

Sintaxis



Descripción

modificadores-Java
Especifica modificadores que son válidos para declaraciones de clases Java, tales como static, public, private o protected.

nombre-clase-Java
Especifica un identificador Java válido. Durante el proceso de preparación del programa, SQLJ genera una clase de contexto de conexión cuyo nombre es este identificador.

cláusula-implements
Consulte "Cláusula implements de SQLJ" para obtener una descripción de esta cláusula. En una cláusula de declaración de conexión, la clase de interfaz referida por la cláusula implements debe ser una clase de interfaz definida por el usuario.

cláusula-with
Consulte "Cláusula with de SQLJ" para obtener una descripción de esta cláusula.

Notas de uso

- SQLJ genera una declaración de clase de conexión para cada cláusula de declaración de conexión especificada por el usuario. Las conexiones con una fuente de datos de SQLJ son objetos de esas clases de conexión generadas.

- Puede especificar una cláusula de declaración de conexión en cualquier lugar de un programa Java en el que pueda existir una definición de clase Java.

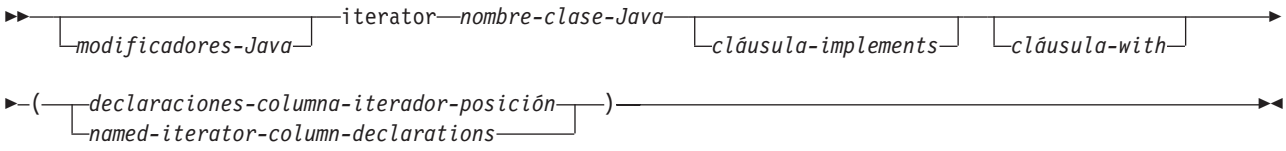
Cláusula de declaración de iterador de SQLJ

Una cláusula de declaración de iterador declara una clase de iterador de posición o de iterador de nombre dentro de un programa de aplicación SQLJ.

Un iterador contiene la tabla de resultados de una consulta. SQLJ genera una clase de iterador para cada cláusula de declaración de iterador especificada por el usuario. Un iterador es un objeto de una clase de iterador.

Una cláusula de declaración de iterador tiene formatos diferentes para un iterador de posición y un iterador de nombre. Las dos clases de iteradores son tipos Java diferentes e incompatibles que se implementan con interfaces diferentes.

Sintaxis



Declaraciones de columna-iterador-posición:

Declaraciones de columna-iterador-nombre:

Descripción

modificadores-Java
Cualquier modificador que sea válido para declaraciones de clases Java, tales como static, public, private o protected.

nombre-clase-Java
Cualquier identificador Java válido. Durante el proceso de preparación del programa, SQLJ genera una clase de iterador cuyo nombre es este identificador.

cláusula-implements
Consulte "Cláusula implements de SQLJ" para obtener una descripción de esta cláusula. Para una cláusula de declaración de iterador que declara un iterador para una operación UPDATE o DELETE de posición, la cláusula implements debe especificar la interfaz sqlj.runtime.ForUpdate. Para una cláusula de declaración de iterador que declara un iterador desplazable, la cláusula implements debe especificar la interfaz sqlj.runtime.Scrollable.

cláusula-with
Consulte "Cláusula with de SQLJ" para obtener una descripción de esta cláusula.

declaraciones-columna-iterador-posición
Especifica una lista de tipos de datos Java, que son los tipos de datos de las columnas del iterador de posición. Los tipos de datos contenidos en la lista deben estar separados por comas. El orden de los tipos de datos en la declaración de iterador de posición es el mismo que el orden de las columnas en la tabla de resultados. Para que sea efectiva la comprobación en línea durante la personalización del perfil serializado, los tipos de datos de las columnas del iterador deben ser compatibles con los tipos de datos de las columnas de la tabla de resultados. Consulte "Tipos de datos Java, JDBC y SQL" para obtener una lista de tipos de datos compatibles.

declaraciones-columna-iterador-nombre
Especifica una lista de tipos de datos Java e identificadores Java, que son los tipos de datos y nombres de las columnas del iterador de nombre. Los pares tipo de datos-nombre deben estar separados por comas. El nombre de una columna del iterador debe coincidir con el nombre de una columna de la tabla de resultados, excepto en lo que respecta al uso de letras mayúsculas y minúsculas. Para que sea efectiva la comprobación en línea durante la personalización del perfil serializado, los tipos de datos de las columnas del iterador deben ser compatibles con los tipos de datos de las columnas de la tabla de resultados. Consulte "Tipos de datos Java, JDBC y SQL" para obtener una lista de tipos de datos compatibles.

Notas de uso

- Una cláusula de declaración de iterador puede aparecer en cualquier punto de un programa Java en donde pueda existir una declaración de clase Java.
- Cuando una declaración de iterador de nombre contiene más de un par de tipos de datos Java e ID de Java, todos los ID de Java de la lista tienen que ser exclusivos. Dos ID de Java no son exclusivos si sólo se diferencian por estar en mayúsculas o minúsculas.

Cláusula ejecutable de SQLJ

Una cláusula ejecutable contiene una sentencia de SQL o una sentencia de asignación. Una sentencia de asignación asigna el resultado de una operación de SQL a una variable Java.

El presente tema describe el formato general de una cláusula ejecutable.

Sintaxis

Notas de uso

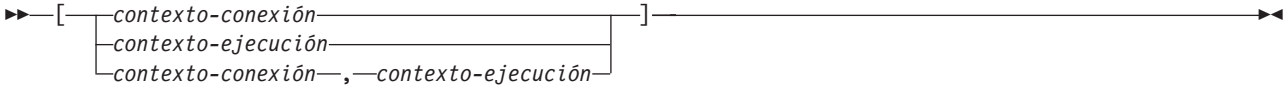
- Una cláusula ejecutable puede aparecer en cualquier lugar de un programa Java en que pueda aparecer una sentencia Java.
- SQLJ utiliza la clase java.sql.SQLException para notificar los códigos negativos de SQL resultantes de cláusulas ejecutables.

Si SQLJ emite una excepción de ejecución durante la ejecución de una cláusula ejecutable, el valor de cualquier expresión de lenguaje principal de tipo OUT o INOUT no está definido.

Cláusula de contexto de SQLJ

La cláusula de contexto especifica un contexto de conexión, un contexto de ejecución o ambas cosas. El contexto de conexión se utiliza para conectar con una fuente de datos. El contexto de ejecución se utiliza para supervisar y modificar la ejecución de sentencias de SQL.

Sintaxis



Descripción

contexto-conexión
Especifica un identificador Java válido que se ha declarado anteriormente en el programa de SQLJ. Este identificador debe estar declarado como instancia de la clase de contexto de conexión que SQLJ genera para una cláusula de declaración de conexión.

contexto-ejecución
Especifica un identificador Java válido que se ha declarado anteriormente en el programa de SQLJ. Este identificador debe estar declarado como instancia de la clase sqlj.runtime.ExecutionContext.

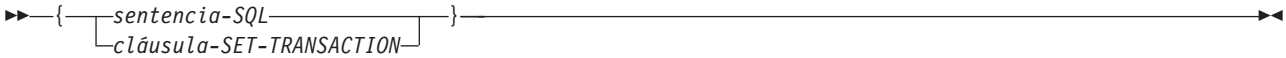
Notas de uso

- Si no especifica un contexto de conexión en una cláusula ejecutable, SQLJ utiliza el contexto de conexión por omisión.
- Si no especifica un contexto de ejecución, SQLJ obtiene el contexto de ejecución a partir del contexto de conexión de la sentencia.

Cláusula de sentencia de SQLJ

Una cláusula de sentencia contiene una sentencia de SQL o una cláusula SET TRANSACTION.

Sintaxis



Descripción

sentencia-SQL
Puede incluir sentencias de SQL en Tabla 105 en la página 462 dentro de una cláusula de sentencia,

cláusula-SET-TRANSACTION
Define el nivel de aislamiento de las sentencias de SQL del programa y la modalidad de acceso de la conexión. La cláusula SET TRANSACTION es

equivalente a la sentencia SET TRANSACTION, que está descrita en el estándar de ANSI/ISO para SQL de 1992 y está soportada en algunas implementaciones de SQL.

Tabla 105. Sentencias de SQL válidas en una cláusula de sentencia de SQLJ

Sentencia	Fuentes de datos aplicables
ALTER DATABASE	1 en la página 464, 2 en la página 464
ALTER FUNCTION	1 en la página 464, 2 en la página 464, 3 en la página 464
ALTER INDEX	1 en la página 464, 2 en la página 464, 3 en la página 464
ALTER PROCEDURE	1 en la página 464, 2 en la página 464, 3 en la página 464
ALTER STOGROUP	1 en la página 464, 2 en la página 464
ALTER TABLE	1 en la página 464, 2 en la página 464, 3 en la página 464
ALTER TABLESPACE	1 en la página 464, 2 en la página 464
CALL	1 en la página 464, 2 en la página 464, 3 en la página 464
COMMENT ON	1 en la página 464, 2 en la página 464
COMMIT	1 en la página 464, 2 en la página 464, 3 en la página 464
SQL compuesto (BEGIN ATOMIC...END)	2 en la página 464
CREATE ALIAS	1 en la página 464, 2 en la página 464
CREATE DATABASE	1 en la página 464, 2 en la página 464, 3a en la página 464
CREATE DISTINCT TYPE	1 en la página 464, 2 en la página 464, 3 en la página 464
CREATE FUNCTION	1 en la página 464, 2 en la página 464, 3 en la página 464
CREATE GLOBAL TEMPORARY TABLE	1 en la página 464, 2 en la página 464
CREATE TEMP TABLE	3 en la página 464
CREATE INDEX	1 en la página 464, 2 en la página 464, 3 en la página 464
CREATE PROCEDURE	1 en la página 464, 2 en la página 464, 3 en la página 464
CREATE STOGROUP	1 en la página 464, 2 en la página 464
CREATE SYNONYM	1 en la página 464, 2 en la página 464, 3 en la página 464
CREATE TABLE	1 en la página 464, 2 en la página 464, 3 en la página 464
CREATE TABLESPACE	1 en la página 464, 2 en la página 464
CREATE TYPE (cursor)	2 en la página 464
CREATE TRIGGER	1 en la página 464, 2 en la página 464, 3 en la página 464
CREATE VIEW	1 en la página 464, 2 en la página 464, 3 en la página 464
DECLARE GLOBAL TEMPORARY TABLE	1 en la página 464, 2 en la página 464
DELETE	1 en la página 464, 2 en la página 464, 3 en la página 464
DROP ALIAS	1 en la página 464, 2 en la página 464
DROP DATABASE	1 en la página 464, 2 en la página 464, 3a en la página 464
DROP DISTINCT TYPE	1 en la página 464, 2 en la página 464
DROP TYPE	3 en la página 464
DROP FUNCTION	1 en la página 464, 2 en la página 464, 3 en la página 464
DROP INDEX	1 en la página 464, 2 en la página 464, 3 en la página 464
DROP PACKAGE	1 en la página 464, 2 en la página 464
DROP PROCEDURE	1 en la página 464, 2 en la página 464, 3 en la página 464
DROP STOGROUP	1 en la página 464, 2 en la página 464

Tabla 105. Sentencias de SQL válidas en una cláusula de sentencia de SQLJ (continuación)

Sentencia	Fuentes de datos aplicables
DROP SYNONYM	1 en la página 464, 2 en la página 464, 3 en la página 464
DROP TABLE	1 en la página 464, 2 en la página 464, 3 en la página 464
DROP TABLESPACE	1 en la página 464, 2 en la página 464
DROP TRIGGER	1 en la página 464, 2 en la página 464, 3 en la página 464
DROP VIEW	1 en la página 464, 2 en la página 464, 3 en la página 464
FETCH	1 en la página 464, 2 en la página 464, 3 en la página 464
GRANT	1 en la página 464, 2 en la página 464, 3 en la página 464
INSERT	1 en la página 464, 2 en la página 464, 3 en la página 464
LOCK TABLE	1 en la página 464, 2 en la página 464, 3 en la página 464
MERGE	1 en la página 464, 2 en la página 464
REVOKE	1 en la página 464, 2 en la página 464, 3 en la página 464
ROLLBACK	1 en la página 464, 2 en la página 464, 3 en la página 464
SAVEPOINT	1 en la página 464, 2 en la página 464, 3 en la página 464
SELECT INTO	1 en la página 464, 2 en la página 464, 3 en la página 464
SET CURRENT APPLICATION ENCODING SCHEME	1 en la página 464
SET CURRENT DEBUG MODE	1 en la página 464
SET CURRENT DEFAULT TRANSFORM GROUP	2 en la página 464
SET CURRENT DEGREE	1 en la página 464, 2 en la página 464
SET CURRENT EXPLAIN MODE	2 en la página 464
SET CURRENT EXPLAIN SNAPSHOT	2 en la página 464
SET CURRENT ISOLATION	1 en la página 464, 2 en la página 464
SET CURRENT LOCALE LC_CTYPE	1 en la página 464
SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION	1 en la página 464, 2 en la página 464
SET CURRENT OPTIMIZATION HINT	1 en la página 464, 2 en la página 464
SET CURRENT PACKAGE PATH	1 en la página 464
SET CURRENT PACKAGESET (no se soporta USER)	1 en la página 464, 2 en la página 464
SET CURRENT PRECISION	1 en la página 464, 2 en la página 464
SET CURRENT QUERY ACCELERATION	1 en la página 464
SET CURRENT QUERY OPTIMIZATION	2 en la página 464
SET CURRENT REFRESH AGE	1 en la página 464, 2 en la página 464
SET CURRENT ROUTINE VERSION	1 en la página 464
SET CURRENT RULES	1 en la página 464
SET CURRENT SCHEMA	2 en la página 464
SET CURRENT SQLID	1 en la página 464
SET PATH	1 en la página 464, 2 en la página 464
TRUNCATE	1 en la página 464
UPDATE	1 en la página 464, 2 en la página 464, 3 en la página 464

Tabla 105. Sentencias de SQL válidas en una cláusula de sentencia de SQLJ (continuación)

Sentencia	Fuentes de datos aplicables

Nota: La sentencia de SQL es aplicable a conexiones con las fuentes de datos siguientes:

1. DB2 para z/OS
2. DB2 Database para Linux, UNIX y Windows
3. IBM Informix
   a. IBM Informix, sólo para la base de datos SYSMASTER.

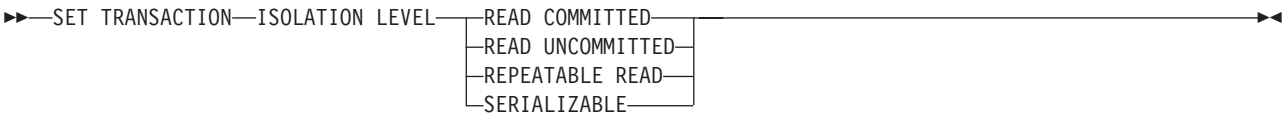
Notas de uso

- SQLJ da soporte a operaciones DELETE y UPDATE de posición y de búsqueda.
- Para una sentencia FETCH, una sentencia DELETE de posición o una sentencia UPDATE de posición, debe utilizar un iterador para apuntar a las filas de una tabla de resultados.

Cláusula SET TRANSACTION de SQLJ

La cláusula SET TRANSACTION define el nivel de aislamiento de la unidad de trabajo actual.

Sintaxis



Descripción

ISOLATION LEVEL
Especifica uno de los niveles de aislamiento siguientes:

READ COMMITTED
Especifica que el nivel de aislamiento actual de DB2 es estabilidad del cursor.

READ UNCOMMITTED
Especifica que el nivel de aislamiento actual de DB2 es lectura no confirmada.

REPEATABLE READ
Especifica que el nivel de aislamiento actual de DB2 es estabilidad de lectura.

SERIALIZABLE
Especifica que el nivel de aislamiento actual de DB2 es lectura repetible.

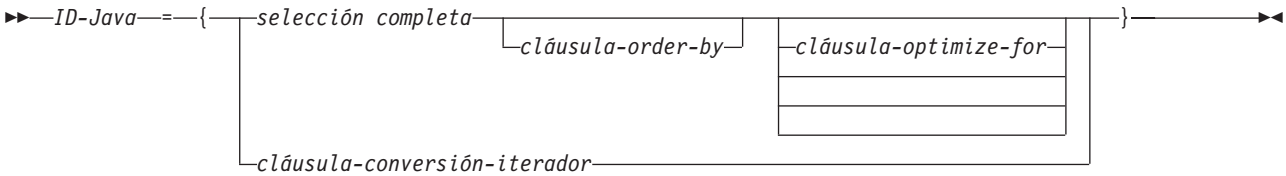
Notas de uso

Puede ejecutar SET TRANSACTION solo al comienzo de una transacción.

Cláusula de asignación de SQLJ

La cláusula de asignación asigna el resultado de una operación de SQL a una variable Java.

Sintaxis



Descripción

ID-Java
Identifica un iterador que se declaró previamente como instancia de una clase de iterador.

selección completa
Genera una tabla de resultados.

cláusula-conversión-iterador
Consulte "Cláusula de conversión de iterador de SQLJ" para obtener una descripción de esta cláusula.

Notas de uso

- Si el objeto identificado por *ID-Java* es un iterador de posición, el número de columnas del conjunto de resultados debe coincidir con el número de columnas del iterador. Además, el tipo de datos de cada columna del conjunto de resultados debe ser compatible con el tipo de datos de la columna correspondiente del iterador. Consulte "Tipos de datos Java, JDBC y SQL para obtener una lista de tipos de datos Java y SQL compatibles.
- Si el objeto identificado por *ID-Java* es un iterador con nombre, el nombre de cada método accesor debe coincidir con el nombre de una columna del conjunto de resultados, salvo en lo que respecta al uso de letras mayúsculas y minúsculas. Además, el tipo de datos del objeto devuelto por un método accesor debe ser compatible con el tipo de datos de la columna correspondiente del conjunto de resultados.
- Puede colocar una cláusula de asignación en cualquier lugar de un programa Java donde pueda aparecer una sentencia de asignación de Java. Sin embargo, no puede poner una cláusula de asignación donde pueda aparecer una expresión de asignación de Java. Por ejemplo, no puede especificar una cláusula de asignación en la lista de control de una sentencia FOR.

Cláusula de conversión a iterador de SQLJ

La cláusula de conversión a iterador convierte un ResultSet de JDBC en un iterador.

Sintaxis

Descripción

expresión-lenguaje-principal
Identifica el ResultSet de JDBC que se debe convertir en un iterador de SQLJ.

Notas de uso

- Si el iterador al que se debe convertir el ResultSet es un iterador de posición, el número de columnas del ResultSet debe coincidir con el número de columnas del iterador. Además, el tipo de datos de cada columna del ResultSet debe ser compatible con el tipo de datos de la columna correspondiente del iterador.
- Si el iterador es un iterador de nombre, el nombre de cada método accesor debe coincidir con el nombre de una columna del ResultSet, salvo en lo que respecta al uso de letras mayúsculas y minúsculas. Además, el tipo de datos del objeto devuelto por un método accesor debe ser compatible con el tipo de datos de la columna correspondiente del ResultSet.
- Cuando se cierra un iterador que se ha generado mediante una cláusula de conversión a iterador, también se cierra el ResultSet a partir del cual se creó el iterador.

Interfaces y clases contenidas en el paquete sqlj.runtime

El paquete sqlj.runtime define las clases de tiempo de ejecución e interfaces utilizadas directa o indirectamente por el programador de SQLJ.

Las clases de tipo AsciiStream son utilizadas directamente por el programador de SQLJ. Las interfaces de tipo ResultSetIterator se implementan como parte de las declaraciones de clase generadas.

Interfaces de sqlj.runtime

La tabla siguiente resume las interfaces contenidas en sqlj.runtime.

Tabla 106. Resumen de las interfaces de sqlj.runtime

Nombre de interfaz	Finalidad
ConnectionContext	Gestiona las operaciones SQL que se realizan durante una conexión a una fuente de datos.
ForUpdate	Se implementa mediante los operadores que se utilizan en sentencias UPDATE o DELETE de posición.
NamedIterator	Se implementa mediante los iteradores declarados como iteradores de nombre.
PositionedIterator	Se implementa mediante los iteradores declarados como iteradores de posición.
ResultSetIterator	Se implementa mediante todos los iteradores para permitir que los resultados de la consulta se procesen utilizando un ResultSet de JDBC.
Scrollable	Proporciona un conjunto de métodos para la manipulación de iteradores desplazables.

Clases de sqlj.runtime

La tabla siguiente resume las clases contenidas en sqlj.runtime.

Tabla 107. Resumen de las clases de sqlj.runtime

Nombre de clase	Finalidad
AsciiStream	Clase para manejar una corriente de entrada cuyos bytes deberían interpretarse como ASCII.
BinaryStream	Clase para manejar una corriente de entrada cuyos bytes deberían interpretarse como binarios.
CharacterStream	Clase para manejar una corriente de entrada cuyos bytes deberían interpretarse como de tipo carácter.

Tabla 107. Resumen de las clases de sqlj.runtime (continuación)

Nombre de clase	Finalidad
DefaultRuntime	Se implementa mediante SQLJ para satisfacer el comportamiento del tiempo de ejecución previsto de SQLJ para la mayoría de los entornos de JVM. Esta clase se utiliza sólo de modo interno y no se describe en esta documentación.
ExecutionContext	Se implementa cuando se declara un contexto de ejecución de SQLJ para controlar la ejecución de operaciones SQL.
Indicador	Define constantes para los valores de variable de indicador.
RuntimeContext	Define los servicios específicos del sistema proporcionados por el entorno de ejecución. Esta clase se utiliza sólo de modo interno y no se describe en esta documentación.
SQLNullException	Deriva de la clase java.sql.SQLException. Se emite una excepción sqlj.runtime.SQLNullException cuando se recupera un valor NULL de SQL en un identificador de sistema principal con un tipo primitivo Java.
StreamWrapper	Deriva una instancia de java.io.InputStream.
UnicodeStream	Clase para manejar una corriente de datos cuyos bytes deberían interpretarse como Unicode.

Interfaz sqlj.runtime.ConnectionContext

La interfaz sqlj.runtime.ConnectionContext proporciona un conjunto de métodos que gestionan las operaciones de SQL que se llevan a cabo durante una sesión con una fuente de datos determinada.

La conversión de una cláusula de declaración de conexión SQLJ hace que SQLJ cree una clase de contexto de conexión. Un objeto de contexto de conexión mantiene un objeto Connection de JDBC en el que se pueden realizar operaciones de SQL dinámico. Un objeto de contexto de conexión también mantiene un objeto ExecutionContext por omisión.

Variables

CLOSE_CONNECTION

Formato:

```
public static final boolean CLOSE_CONNECTION=true;
```

Constante que puede pasarse al método close. Indica que el objeto de conexión de JDB subyacente debe cerrarse.

KEEP_CONNECTION

Formato:

```
public static final boolean KEEP_CONNECTION=false;
```

Constante que puede pasarse al método close. Indica que el objeto de conexión de JDB subyacente no debe cerrarse.

Métodos

close()

Formato:

```
public abstract void close() throws SQLException
```

Realiza las funciones siguientes:

- Libera todos los recursos utilizados por el objeto de contexto de conexión dado.
- Cierra todos los objetos ConnectedProfile abiertos.
- Cierra el objeto Connection de JDBC subyacente.

close() es equivalente a close(CLOSE_CONNECTION).

close(boolean)

Formato:

```
public abstract void close (boolean
cerrar-conexión)
 throws SQLException
```

Realiza las funciones siguientes:
- Libera todos los recursos utilizados por el objeto de contexto de conexión dado.
- Cierra todos los objetos ConnectedProfile abiertos.
- Cierra el objeto Connection de JDBC subyacente, en función del valor del parámetro *cerrar-conexión*.

Parámetros:

cerrar-conexión
Especifica si el objeto Connection de JDBC se cierra cuando se cierra un objeto de contexto de conexión:

CLOSE_CONNECTION
Cierra el objeto Connection de JDBC subyacente.

KEEP_CONNECTION
No cierra el objeto Connection de JDBC subyacente.

getConnectedProfile

Formato:

```
public abstract ConnectedProfile getConnectedProfile(Object profileKey)
 throws SQLException
```

Este método lo utiliza código que se genera mediante el conversor SQLJ. No está indicado para el uso directo por parte de los programas de aplicación.

getConnection

Formato:

```
public abstract Connection getConnection()
```

Devuelve el objeto Connection de JDBC subyacente correspondiente al objeto de contexto de conexión dado.

getExecutionContext

Formato:

```
public abstract ExecutionContext getExecutionContect()
```

Devuelve el objeto ExecutionContext por omisión asociado con el objeto de contexto de conexión dado.

isClosed

Formato:

```
public abstract boolean isClosed()
```

Devuelve `true` si el objeto de contexto de conexión dado se ha cerrado. Devuelve `false` si el objeto de contexto de conexión no se ha cerrado.

Constructores

Los constructores siguientes se definen en una implementación concreta de la interfaz ConnectionContext que resulta de la conversión de la sentencia `#sql context Ctx;`:

Ctx(String, boolean)

Formato:

```
public Ctx(String url, boolean confirmación-automática)
 throws SQLException
```

Parámetros:

url
: Representación de una fuente de datos, tal como se ha especificado en el método getConnection de JDBC.

confirmación-automática
: Indica si se ha habilitado la confirmación automática para la conexión. El valor `true` significa que la confirmación automática está habilitada. El valor `false` significa que la confirmación automática está inhabilitada.

Ctx(String, String, String, boolean)

Formato:

```
public Ctx(String url, String usuario, String contraseña,
 boolean confirmación-automática)
 throws SQLException
```

Parámetros:

url
: Representación de una fuente de datos, tal como se ha especificado en el método getConnection de JDBC.

user
: ID de usuario con el que se ha establecido la conexión con la fuente de datos.

contraseña
: Contraseña del ID de usuario con el que se ha establecido la conexión con la fuente de datos.

confirmación-automática
: Indica si se ha habilitado la confirmación automática para la conexión. El valor `true` significa que la confirmación automática está habilitada. El valor `false` significa que la confirmación automática está inhabilitada.

Ctx(String, Properties, boolean)

Formato:

```
public Ctx(String url, Properties info, boolean
confirmación-automática)
 throws SQLException
```

Parámetros:

url
: Representación de una fuente de datos, tal como se ha especificado en el método getConnection de JDBC.

info
: Objeto de tipo que contiene un conjunto de propiedades de controlador

para la conexión. Se puede especificar cualquiera de las propiedades de IBM Data Server Driver para JDBC y SQLJ.

confirmación-automática
Indica si se ha habilitado la confirmación automática para la conexión. El valor `true` significa que la confirmación automática está habilitada. El valor `false` significa que la confirmación automática está inhabilitada.

Ctx(Connection)
Formato:

```
public Ctx(java.sql.Connection objeto-conexión-JDBC)
 throws SQLException
```

Parámetros:

objeto-conexión-JDBC
Objeto Connection de JDBC creado previamente.

Si la llamada del constructor emite una excepción de SQL, el objeto Connection de JDBC permanece abierto.

Ctx(ConnectionContext)
Formato:

```
public Ctx(sqlj.runtime.ConnectionContext
objeto-contexto-conexión-SQLJ)
 throws SQLException
```

Parámetros:

objeto-contexto-conexión-SQLJ
Objeto ConnectionContext de SQLJ creado previamente.

Los constructores siguientes se definen en una implementación concreta de la interfaz ConnectionContext que resulta de la conversión de la sentencia `#sql context Ctx with (dataSource ="jdbc/TestDS");`:

Ctx()
Formato:

```
public Ctx()
 throws SQLException
```

Ctx(String, String)
Formato:

```
public Ctx(String usuario, String
contraseña,
)
 throws SQLException
```

Parámetros:

user
ID de usuario con el que se ha establecido la conexión con la fuente de datos.

contraseña
Contraseña del ID de usuario con el que se ha establecido la conexión con la fuente de datos.

Ctx(Connection)
Formato:

```
public Ctx(java.sql.Connection objeto-conexión-JDBC)
 throws SQLException
```

Parámetros:

objeto-conexión-JDBC
Objeto Connection de JDBC creado previamente.

Si la llamada del constructor emite una excepción de SQL, el objeto Connection de JDBC permanece abierto.

Ctx(ConnectionContext)

Formato:

```
public Ctx(sqlj.runtime.ConnectionContext
objeto-contexto-conexión-SQLJ)
 throws SQLException
```

Parámetros:

objeto-contexto-conexión-SQLJ
Objeto ConnectionContext de SQLJ creado previamente.

Métodos

Los métodos adicionales siguientes se crean en una implementación concreta de la interfaz ConnectionContext que resulta de la conversión de la sentencia `#sql context Ctx;`:

getDefaultContext

Formato:

```
public static Ctx getDefaultContext()
```

Devuelve el objeto del contexto de conexión por omisión correspondiente a la clase Ctx.

getProfileKey

Formato:

```
public static Object getProfileKey(sqlj.runtime.profile.Loader cargador,
String nombre-perfil) throws SQLException
```

Este método lo utiliza código que se genera mediante el conversor SQLJ. No está indicado para el uso directo por parte de los programas de aplicación.

getProfile

Formato:

```
public static sqlj.runtime.profile.Profile getProfile(Object clave)
```

Este método lo utiliza código que se genera mediante el conversor SQLJ. No está indicado para el uso directo por parte de los programas de aplicación.

getTypeMap

Formato:

```
public static java.util.Map getTypeMap()
```

Devuelve una instancia de una clase que implementa java.util.Map, que es la correlación de tipos definidos por el usuario asociada con ConnectionContext. Si no existe una correlación de tipos asociada, se devuelve un nulo Java.

Este método lo utiliza código que se genera mediante el conversor SQLJ para las cláusulas ejecutables y de declaración de iterador, pero también se pueden invocar en una aplicación SQLJ para su uso directo en sentencias JDBC.

SetDefaultContext

Formato:

```
public static void Ctx setDefaultContext(Ctx contexto-por-omisión)
```

Define el objeto del contexto de conexión por omisión correspondiente a la clase Ctx.

Recomendación: no utilice este método para las aplicaciones de varias hebras. En su lugar, utilice contextos explícitos.

Interfaz sqlj.runtime.ForUpdate

SQLJ implementa la interfaz sqlj.runtime.ForUpdate en los programas SQLJ que contienen una cláusula de declaración del iterador con `implements sqlj.runtime.ForUpdate`.

Un programa SQLJ que efectúa operaciones UPDATE o DELETE de posición (UPDATE...WHERE CURRENT OF o DELETE...WHERE CURRENT OF) debe incluir una cláusula de declaración del iterador con `implements sqlj.runtime.ForUpdate`.

Métodos

getCursorName

Formato:

```
public abstract String getCursorName() throws SQLException
```

Este método lo utiliza código que se genera mediante el conversor SQLJ. No está indicado para el uso directo por parte de los programas de aplicación.

Interfaz sqlj.runtime.NamedIterator

La interfaz sqlj.runtime.NamedIterator se implementa cuando una aplicación SQLJ ejecuta una cláusula de declaración de iterador para un iterador determinado.

Un iterador de nombres incluye nombres de columnas de la tabla de resultados; el orden de éstas en el iterador no es importante.

En las implementaciones de la interfaz sqlj.runtime.NamedIterator se incluye un método accesor para cada columna de la tabla de resultados. Un método accesor devuelve los datos contenidos en las columnas de la tabla de resultados. El nombre de un método accesor coincide con el nombre de la columna correspondiente del iterador de nombre.

Métodos (heredados de la interfaz ResultSetIterator)

close

Formato:

```
public abstract void close() throws SQLException
```

Libera recursos de base de datos que el iterador está utilizando.

isClosed

Formato:

```
public abstract boolean isClosed() throws SQLException
```

Devuelve el valor `true` si se ha invocado el método close. Devuelve el valor `false` si no se ha invocado el método.

next

Formato:

```
public abstract boolean next() throws SQLException
```

Avanza el iterador hasta la fila siguiente. Antes de invocar por primera vez una instancia del método next, el iterador se posiciona delante de la primera fila de la tabla de resultados. next devuelve el valor `true` cuando hay una fila siguiente disponible, y devuelve `false` cuando se han recuperado todas las filas.

Interfaz sqlj.runtime.PositionedIterator

La interfaz sqlj.runtime.PositionedIterator se implementa cuando una aplicación SQLJ ejecuta una cláusula de declaración de iterador para un iterador de posición.

El orden de las columnas de un iterador de posición debe ser el mismo que el orden de las columnas de la tabla de resultados, y un iterador de posición no incluye nombres de columna de tabla de resultados.

Métodos

sqlj.runtime.PositionedIterator hereda todos los métodos **ResultSetIterator**, e incluye el método adicional siguiente:

endFetch

Formato:

```
public abstract boolean endFetch() throws SQLException
```

Devuelve el valor `true` si el iterador no está posicionado en una fila. Devuelve el valor `false` si el iterador está posicionado en una fila.

Interfaz sqlj.runtime.ResultSetIterator

SQLJ implementa la interfaz sqlj.runtime.ResultSetIterator para todas las cláusulas de declaración de iterador.

Un iterador sin tipo se puede generar mediante la declaración de una instancia de la interfaz sqlj.runtime.ResultSetIterator directamente. En general, el uso de iteradores sin tipo no es recomendable.

Variables

ASENSITIVE

Formato:

```
public static final int ASENSITIVE
```

Constante que el método getSensitivity puede devolver. Indica que el iterador está definido como ASENSITIVE.

Este valor no es devuelto por IBM Informix.

FETCH_FORWARD

Formato:

```
public static final int FETCH_FORWARD
```

Constante que los métodos siguientes pueden utilizar:

- Establecida por sqlj.runtime.Scrollable.setFetchDirection y sqlj.runtime.ExecutionContext.setFetchDirection
- Devuelta por sqlj.runtime.ExecutionContext.getFetchDirection

Indica que el iterador capta filas en una tabla de resultados hacia adelante, desde la primera hasta la última.

FETCH_REVERSE

Formato:

```
public static final int FETCH_REVERSE
```

Constante que los métodos siguientes pueden utilizar:
- Establecida por sqlj.runtime.Scrollable.setFetchDirection y sqlj.runtime.ExecutionContext.setFetchDirection
- Devuelta por sqlj.runtime.ExecutionContext.getFetchDirection

Indica que el iterador capta filas en una tabla de resultados hacia atrás, desde la última hasta la primera.

Este valor no es devuelto por IBM Informix.

FETCH_UNKNOWN

Formato:

```
public static final int FETCH_UNKNOWN
```

Constante que los métodos siguientes pueden utilizar:
- Establecida por sqlj.runtime.Scrollable.setFetchDirection y sqlj.runtime.ExecutionContext.setFetchDirection
- Devuelta por sqlj.runtime.ExecutionContext.getFetchDirection

Indica que el iterador capta filas en una tabla de resultados en un orden desconocido.

Este valor no es devuelto por IBM Informix.

INSENSITIVE

Formato:

```
public static final int INSENSITIVE
```

Constante que el método getSensitivity puede devolver. Indica que el iterador está definido como INSENSITIVE.

SENSITIVE

Formato:

```
public static final int SENSITIVE
```

Constante que el método getSensitivity puede devolver. Indica que el iterador está definido como SENSITIVE.

Este valor no es devuelto por IBM Informix.

Métodos

clearWarnings

Formato:

```
public abstract void clearWarnings() throws SQLException
```

Después de llamar a clearWarnings, getWarnings devuelve un valor nulo hasta que se informa de un nuevo aviso para el iterador.

close

Formato:

```
public abstract void close() throws SQLException
```

Cierra el iterador y libera los recursos de base de datos subyacentes.

getFetchSize

Formato:

```
synchronized public int getFetchSize() throws SQLException
```

Devuelve el número de filas que SQLJ debería captar cuando se necesita más de una fila. El valor devuelto es aquél que se definió con el método setFetchSize, o 0 si no se definió ningún valor en setFetchSize.

getResultSet

Formato:

```
public abstract ResultSet getResultSet() throws SQLException
```

Devuelve el objeto ResultSet de JDBC asociado con el iterador.

getRow

Formato:

```
synchronized public int getRow() throws SQLException
```

Devuelve el número de fila actual. La primera fila corresponde al número 1, la segunda corresponde al número 2, etc. Si el iterador no se encuentra en una fila, se devuelve 0.

getSensitivity

Formato:

```
synchronized public int getSensitivity() throws SQLException
```

Devuelve la sensibilidad del iterador. La sensibilidad se determina mediante el valor de sensibilidad que se ha especificado en la cláusula with de la cláusula de declaración de iterador o mediante el valor por omisión de dicha cláusula with.

getWarnings

Formato:

```
public abstract SQLWarning getWarnings() throws SQLException
```

Devuelve el primer aviso notificado por las llamadas en el iterador. Los avisos del iterador subsiguientes se encadenan a este aviso de SQL. La serie de avisos se borra automáticamente cada vez que el iterador se desplaza a una fila nueva.

isClosed

Formato:

```
public abstract boolean isClosed() throws SQLException
```

Devuelve el valor `true` si el iterador está cerrado. En caso contrario devuelve `false`.

next

Formato:

```
public abstract boolean next() throws SQLException
```

Avanza el iterador hasta la fila siguiente. Antes de invocar next por primera vez, el iterador se coloca delante de la primera fila de la tabla de resultados. next devuelve el valor `true` cuando hay una fila siguiente disponible, y devuelve `false` cuando se han recuperado todas las filas.

setFetchSize

Formato:

```
synchronized public void setFetchSize(int número de filas) throws SQLException
```

Proporciona a SQLJ una pista sobre el número de filas que deben captarse cuando se necesitan más filas.

Parámetros:

número de filas
El número esperado de filas que SQLJ debe captar para el iterador asociado al contexto de ejecución definido.

Si *número de filas* es menor que 0 o mayor que el número máximo de filas que pueden captarse, se emite una excepción de SQL (SQLException).

Interfaz sqlj.runtime.Scrollable

sqlj.runtime.Scrollable proporciona métodos para moverse por la tabla de resultados y para comprobar la posición de dicha tabla.

sqlj.runtime.Scrollable se implementa cuando se declara un iterador desplazable.

Métodos

absolute(int)

Formato:

```
public abstract boolean absolute (int
n) throws SQLException
```

Desplaza el cursor hasta una fila especificada.

Si *n*>0, sitúa el iterador en la fila *n* de la tabla de resultados. Si *n*<0 y *m* es el número de filas de la tabla de resultados, sitúa el iterador en la fila *m*+*n*+*1* de la tabla de resultados.

Si el valor absoluto de *n* es mayor que el número de filas de la tabla de resultados, sitúa el cursor después de la última fila si *n* es positivo o antes de la primera fila si *n* es negativo.

absolute(0) es lo mismo que beforeFirst(). absolute(1) es lo mismo que first(). absolute(-1) es lo mismo que last().

Devuelve el valor `true` si el iterador está en una fila. En otro caso, devuelve `false`.

afterLast()

Formato:

```
public abstract void afterLast() throws SQLException
```

Coloca el iterador después de la última fila de la tabla de resultados.

beforeFirst()

Formato:

```
public abstract void beforeFirst() throws SQLException
```

Coloca el iterador antes de la primera fila de la tabla de resultados.

first()

Formato:

```
public abstract boolean first() throws SQLException
```

Mueve el iterador a la primera fila de la tabla de resultados.

Devuelve el valor `true` si el iterador está en una fila. En otro caso, devuelve `false`.

getFetchDirection()

Formato:

```
public abstract int getFetchDirection ( ) throws SQLException
```

Devuelve la dirección de captación del iterador. Los valores posibles son:

sqlj.runtime.ResultSetIterator.FETCH_FORWARD
Las filas se procesan en dirección de avance, de la primera a la última.

sqlj.runtime.ResultSetIterator.FETCH_REVERSE
Las filas se procesan en dirección de retroceso, de la última a la primera.

sqlj.runtime.ResultSetIterator.FETCH_UNKNOWN
El orden del proceso no es conocido.

isAfterLast()

Formato:

```
public abstract boolean isAfterLast() throws SQLException
```

Devuelve el valor `true` si el iterador está situado después de la última fila de la tabla de resultados. En otro caso, devuelve `false`.

isBeforeFirst()

Formato:

```
public abstract boolean isBeforeFirst() throws SQLException
```

Devuelve el valor `true` si el iterador está situado antes de la primera fila de la tabla de resultados. En otro caso, devuelve `false`.

isFirst()

Formato:

```
public abstract boolean isFirst() throws SQLException
```

Devuelve el valor `true` si el iterador está situado en la primera fila de la tabla de resultados. En otro caso, devuelve `false`.

isLast()

Formato:

```
public abstract boolean isLast() throws SQLException
```

Devuelve el valor `true` si el iterador está situado en la última fila de la tabla de resultados. En otro caso, devuelve `false`.

last()

Formato:

```
public abstract boolean last() throws SQLException
```

Mueve el iterador a la última fila de la tabla de resultados.

Devuelve el valor `true` si el iterador está en una fila. En otro caso, devuelve `false`.

previous()

Formato:

```
public abstract boolean previous() throws SQLException
```

Mueve el iterador a la fila anterior de la tabla de resultados.

Devuelve el valor `true` si el iterador está en una fila. En otro caso, devuelve `false`.

relative(int)

Formato:

```
public abstract boolean relative(int
n) throws SQLException
```

Si *n*>0, sitúa el iterador en la fila que está *n* filas después de la fila actual. Si *n*<0, sitúa el iterador en la fila que está situada *n* filas antes de la fila actual. Si *n*=0, sitúa el iterador en la fila actual.

El cursor debe estar en una fila válida de la tabla de resultados para poder utilizar este método. Si el cursor está antes de la primera fila o después de la última fila, el método emite una SQLException.

Suponga que *m* es el número de filas de la tabla de resultados y *x* es el número de fila actual de la tabla de resultados. Si *n*>0 y *x*+*n*>*m*, el iterador se sitúa a continuación de la última fila. Si *n*<0 y *x*+*n*<1, el iterador se sitúa antes de la primera fila.

Devuelve el valor `true` si el iterador está en una fila. En otro caso, devuelve `false`.

setFetchDirection(int)

Formato:

```
public abstract void setFetchDirection (int) throws SQLException
```

Indica al entorno de ejecución de SQLJ la dirección en la que se procesan las filas del objeto iterador. Los valores posibles son:

sqlj.runtime.ResultSetIterator.FETCH_FORWARD
: Las filas se procesan en dirección de avance, de la primera a la última.

sqlj.runtime.ResultSetIterator.FETCH_REVERSE
: Las filas se procesan en dirección de retroceso, de la última a la primera.

sqlj.runtime.ResultSetIterator.FETCH_UNKNOWN
: El orden del proceso no es conocido.

Clase sqlj.runtime.AsciiStream

La clase sqlj.runtime.AsciiStream se utiliza para una corriente de entrada de datos ASCII con una longitud específica.

La clase sqlj.runtime.AsciiStream deriva de la clase java.io.InputStream y amplía la clase sqlj.runtime.StreamWrapper. SQLJ interpreta los bytes de un objeto sqlj.runtime.AsciiStream como caracteres ASCII. Los objetos InputStream con caracteres ASCII deben pasarse como objetos sqlj.runtime.AsciiStream.

Constructores

AsciiStream(InputStream)

Formato:

```
public AsciiStream(java.io.InputStream corriente-entrada)
```

Crea un objeto java.io.InputStream ASCII con una longitud no especificada.

Parámetros:

corriente-entrada
Objeto InputStream que SQLJ interpreta como objeto AsciiStream .

AsciiStream(InputStream, int)

Formato:

```
public AsciiStream(java.io.InputStream corriente-entrada, int longitud)
```

Crea un objeto java.io.InputStream ASCII con una longitud especificada.

Parámetros:

corriente-entrada
Objeto InputStream que SQLJ interpreta como objeto AsciiStream .

longitud
Longitud del objeto InputStream que SQLJ interpreta como objeto AsciiStream.

Clase sqlj.runtime.BinaryStream

La clase sqlj.runtime.BinaryStream se utiliza para una corriente de entrada de datos binarios con una longitud específica.

La clase sqlj.runtime.BinaryStream deriva de la clase java.io.InputStream y amplía la clase sqlj.runtime.StreamWrapper. SQLJ interpreta los bytes de un objeto sqlj.runtime.BinaryStream como caracteres binarios. Los objetos InputStream con caracteres binarios deben pasarse como objetos sqlj.runtime.BinaryStream.

Constructores

BinaryStream(InputStream)

Formato:

```
public BinaryStream(java.io.InputStream corriente-entrada)
```

Crea un objeto java.io.InputStream binario con una longitud no especificada.

Parámetros:

corriente-entrada
Objeto InputStream que SQLJ interpreta como objeto BinaryStream .

BinaryStream(InputStream, int)

Formato:

```
public BinaryStream(java.io.InputStream corriente-entrada, int longitud)
```

Crea un objeto java.io.InputStream binario con una longitud especificada.

Parámetros:

corriente-entrada
Objeto InputStream que SQLJ interpreta como objeto BinaryStream .

longitud
Longitud del objeto InputStream que SQLJ interpreta como objeto BinaryStream.

Clase sqlj.runtime.CharacterStream

La clase sqlj.runtime.CharacterStream se utiliza para la corriente de entrada de datos de tipo carácter con una longitud especificada.

La clase sqlj.runtime.CharacterStream se deriva de la clase java.io.Reader y amplía la clase java.io.FilterReader. SQLJ interpreta los bytes de un objeto sqlj.runtime.CharacterStream como datos Unicode. Debe pasarse un objeto Reader con datos Unicode como objeto sqlj.runtime.CharacterStream.

Constructores

CharacterStream(InputStream)
Formato:

```
public CharacterStream(java.io.Reader corriente-entrada)
```

Crea un objeto java.io.Reader de tipo carácter con una longitud no especificada.

Parámetros:

corriente-entrada
Objeto Reader que SQLJ interpreta como objeto CharacterStream.

CharacterStream(InputStream, int)
Formato:

```
public CharacterStream(java.io.Reader corriente-entrada, int longitud)
```

Crea un objeto java.io.Reader de tipo carácter con una longitud especificada.

Parámetros:

corriente-entrada
Objeto Reader que SQLJ interpreta como objeto CharacterStream.

longitud
Longitud del objeto Reader que SQLJ interpreta como objeto CharacterStream.

Métodos

getReader
Formato:

```
public Reader getReader()
```

Devuelve el objeto Reader subyacente derivado mediante el objeto CharacterStream.

getLength
Formato:

```
public void getLength()
```

Devuelve la longitud en caracteres del objeto Reader derivado, tal como se ha especificado mediante el constructor o en la última llamada a setLength.

setLength
Formato:

```
public void setLength (int longitud)
```

Establece el número de caracteres que se leen en el objeto Reader cuando el objeto se pasa como argumento de entrada a una operación de SQL.

Parámetros:

longitud
Número de caracteres que se leen en el objeto Reader.

Clase sqlj.runtime.ExecutionContext

La clase sqlj.runtime.ExecutionContext se define para contextos de ejecución. Se utiliza un contexto de ejecución para controlar la ejecución de sentencias de SQL.

Variables

ADD_BATCH_COUNT

Formato:

```
public static final int ADD_BATCH_COUNT
```

Constante que el método getUpdateCount puede devolver. Indica que la sentencia anterior no se ejecutó, pero se añadió al lote de sentencias existente.

AUTO_BATCH

Formato:

```
public static final int AUTO_BATCH
```

Constante que puede pasarse al método setBatchLimit. Indica que debe realizarse la ejecución por lotes implícita, y que SQLJ debe determinar el tamaño del lote.

DBDefault

Formato:

```
public static final short DBDefault=-5;
```

Constante que puede asignarse a una variable de indicador. Especifica que el valor de la variable de lenguaje principal correspondiente que se pasa al servidor de datos es el valor por omisión.

DBNonNull

Formato:

```
public static final short DBNonNull=0;
```

Constante que puede asignarse a una variable de indicador. Especifica que el valor de la variable de lenguaje principal correspondiente que se pasa al servidor de datos es un valor no nulo.

DBNull

Formato:

```
public static final short DBNull=-1;
```

Constante que puede asignarse a una variable de indicador. Especifica que el valor de la variable de lenguaje principal correspondiente que se pasa al servidor de datos es el valor nulo de SQL.

DBUnassigned

Formato:

```
public static final short DBUnassigned=-7;
```

Constante que puede asignarse a una variable de indicador. Especifica que ningún valor para la variable de lenguaje principal correspondiente se pasa al servidor de datos.

EXEC_BATCH_COUNT
Formato:

```
public static final int EXEC_BATCH_COUNT
```

Constante que puede volver del método getUpdateCount. Indica que se acaba de ejecutar un lote de sentencias.

EXCEPTION_COUNT
Formato:

```
public static final int EXCEPTION_COUNT
```

Constante que puede volver del método getUpdateCount. Indica que se ha emitido una excepción antes de que finalizara la ejecución anterior, o que no se ha realizado ninguna operación en el objeto de contexto de ejecución.

NEW_BATCH_COUNT
Formato:

```
public static final int NEW_BATCH_COUNT
```

Constante que puede volver del método getUpdateCount. Indica que la sentencia anterior no se ejecutó, pero se añadió al lote de sentencias nuevo.

QUERY_COUNT
Formato:

```
public static final int QUERY_COUNT
```

Constante que puede pasarse al método setBatchLimit. Indica que la ejecución anterior generó un conjunto de resultados.

UNLIMITED_BATCH
Formato:

```
public static final int UNLIMITED_BATCH
```

Constante que puede volver del método getUpdateCount. Indica que las sentencias deben seguir añadiéndose a un lote de sentencias, independientemente del tamaño del lote.

Constructores:

ExecutionContext
Formato:

```
public ExecutionContext()
```

Crea una instancia ExecutionContext.

Métodos

cancel
Formato:

```
public void cancel() throws SQLException
```

Cancela la operación SQL que actualmente está ejecutando una hebra que utiliza el objeto de contexto de ejecución. Si hay un lote de sentencias pendiente en el objeto de contexto de ejecución, el lote de sentencias se cancela y se borra.

El método cancel emite SQLException si la sentencia no se puede cancelar.

execute

Formato:

```
public boolean execute ( ) throws SQLException
```

Este método lo utiliza código que se genera mediante el conversor SQLJ. No está indicado para el uso directo por parte de los programas de aplicación.

executeBatch

Formato:

```
public synchronized int[] executeBatch() throws SQLException
```

Ejecuta el lote de sentencias pendiente y devuelve una matriz de contajes de actualización. Si no existe ningún lote de sentencias pendiente, se devuelve un valor nulo. Cuando se invoca este método, se elimina el lote de sentencias, aunque la llamada produzca una excepción.

Cada elemento de la matriz devuelta puede ser uno de estos valores:

-2 Este valor indica que la sentencia de SQL se ejecutó satisfactoriamente, pero no se pudo determinar el número de filas que fueron actualizadas.

-3 Este valor indica que la sentencia de SQL falló.

Otro valor entero
Este valor es el número de filas que fueron actualizadas por la sentencia.

El método executeBatch emite un SQLException si se produce un error en la base de datos durante la ejecución del lote de sentencias.

executeQuery

Formato:

```
public RTResultSet executeQuery ( ) throws SQLException
```

Este método lo utiliza código que se genera mediante el conversor SQLJ. No está indicado para el uso directo por parte de los programas de aplicación.

executeUpdate

Formato:

```
public int executeUpdate() throws SQLException
```

Este método lo utiliza código que se genera mediante el conversor SQLJ. No está indicado para el uso directo por parte de los programas de aplicación.

getBatchLimit

Formato:

```
synchronized public int getBatchLimit()
```

Devuelve el número de sentencias que se añaden a un lote antes de ejecutarlo implícitamente.

El valor que se devuelve es uno de los siguientes:

UNLIMITED_BATCH
Este valor indica que el tamaño del lote es ilimitado.

AUTO_BATCH
Este valor indica que el tamaño del lote es limitado pero desconocido.

Otro valor entero
El límite del lote actual.

getBatchUpdateCounts
Formato:

```
public synchronized int[] getBatchUpdateCounts()
```

Devuelve una matriz que contiene el número de filas que cada sentencia actualizó y que se ejecutaron satisfactoriamente en un lote. El orden de los elementos de la matriz corresponde al orden en el que se insertaron las sentencias en el lote. Devuelve un valor nulo si ninguna sentencia del lote se ejecutó satisfactoriamente.

Cada elemento de la matriz devuelta puede ser uno de estos valores:

-2 Este valor indica que la sentencia de SQL se ejecutó satisfactoriamente, pero no se pudo determinar el número de filas que fueron actualizadas.

-3 Este valor indica que la sentencia de SQL falló.

Otro valor entero
Este valor es el número de filas que fueron actualizadas por la sentencia.

getFetchDirection
Formato:

```
synchronized public int getFetchDirection() throws SQLException
```

Devuelve la dirección de captación actual para los objetos del iterador desplazable que se generaron a partir del contexto de ejecución definido. Si en el contexto de ejecución no se definió una dirección de captación, se devuelve sqlj.runtime.ResultSetIterator.FETCH_FORWARD.

getFetchSize
Formato:

```
synchronized public int getFetchSize() throws SQLException
```

Devuelve el número de filas que SQLJ debería captar cuando se necesita más de una fila. Este valor sólo se aplica a los objetos del iterador que se generaron a partir del contexto de ejecución definido. El valor devuelto es aquél que se definió con el método setFetchSize, o 0 si no se definió ningún valor en setFetchSize.

getMaxFieldSize
Formato:

```
public synchronized int getMaxFieldSize()
```

Devuelve el número máximo de bytes que se devuelven para cualquier columna de serie (de caracteres, gráfica o binaria de longitud variable) en consultas que utilizan el contexto de ejecución definido. Si se sobrepasa este límite, SQLJ descarta los bytes restantes. Un valor 0 significa que el número máximo de bytes es ilimitado.

getMaxRows
Formato:

```
public synchronized int getMaxRows()
```

Proporciona el número máximo de filas que son devueltas por una consulta cualquiera que hace uso del contexto de ejecución existente. Si se sobrepasa este límite, SQLJ descarta las filas restantes. Si el valor devuelto es 0, significa que el número máximo de filas es ilimitado.

getNextResultSet()

Formato:

```
public ResultSet getNextResultSet() throws SQLException
```

Después de una llamada de procedimiento almacenado, devuelve un conjunto de resultados procedente del procedimiento almacenado.

Se devuelve un valor nulo si se produce cualquiera de estas situaciones:

- No hay más conjuntos de resultados que devolver.
- La llamada del procedimiento almacenado no generó ningún conjunto de resultados.
- No se ha ejecutado una llamada de procedimiento almacenado en el contexto de ejecución.

Cuando invoca getNextResultSet(), SQLJ cierra el conjunto de resultados que está abierto actualmente y pasa al conjunto de resultados siguiente.

Si se produce un error durante una llamada a getNextResultSet, se liberarán los recursos para el objeto ResultSet de JDBC actual, y se emitirá un SQLException. Las llamadas subsiguientes a getNextResultSet devolverán un valor nulo.

getNextResultSet(int)

Formatos:

```
public ResultSet getNextResultSet(int actual)
```

Después de una llamada de procedimiento almacenado, devuelve un conjunto de resultados procedente del procedimiento almacenado.

Se devuelve un valor nulo si se produce cualquiera de estas situaciones:

- No hay más conjuntos de resultados que devolver.
- La llamada del procedimiento almacenado no generó ningún conjunto de resultados.
- No se ha ejecutado una llamada de procedimiento almacenado en el contexto de ejecución.

Si se produce un error durante una llamada a getNextResultSet, se liberarán los recursos para el objeto ResultSet de JDBC actual, y se emitirá un SQLException. Las llamadas subsiguientes a getNextResultSet devolverán un valor nulo.

Parámetros:

actual
: Indica la acción que lleva a cabo SQLJ con el conjunto de resultados actualmente abierto antes de pasar al siguiente conjunto de resultados.

  java.sql.Statement.CLOSE_CURRENT_RESULT
  : Especifica que el objeto ResultSet actual se cierra cuando se devuelve el objeto ResultSet siguiente.

  java.sql.Statement.KEEP_CURRENT_RESULT
  : Especifica que el objeto ResultSet actual permanece abierto cuando se devuelve el objeto ResultSet siguiente.

java.sql.Statement.CLOSE_ALL_RESULTS
Especifica que todos los objetos ResultSet abiertos se cierran cuando se devuelve el objeto ResultSet siguiente.

getQueryTimeout
Formato:

```
public synchronized int getQueryTimeout()
```

Devuelve el número máximo de segundos que pueden ejecutar las operaciones SQL que utilizan el objeto de contexto de ejecución definido. Si una operación SQL sobrepasa el límite, se emite un SQLException. El valor devuelto es aquél que se definió con el método setQueryTimeout, o 0 si no se definió ningún valor en setQueryTimeout. 0 significa que el tiempo de ejecución es ilimitado.

getUpdateCount
Formato:

```
public abstract int getUpdateCount() throws SQLException
```

Devuelve:

ExecutionContext.ADD_BATCH_COUNT
Si la sentencia se añadió a un lote existente.

ExecutionContext.NEW_BATCH_COUNT
Si la sentencia era la primera sentencia de un nuevo lote.

ExecutionContext.EXCEPTION_COUNT
Si la sentencia anterior generó un SQLException, o no se ejecutó ninguna sentencia anterior.

ExecutionContext.EXEC_BATCH_COUNT
Si la sentencia era parte de un lote y el lote se ejecutó.

ExecutionContext.QUERY_COUNT
Si la sentencia anterior creó un objeto iterador o un ResultSet de JDBC.

Otro valor entero
Si la sentencia se ejecutó en lugar de añadirla a un lote. Este valor es el número de filas que fueron actualizadas por la sentencia.

getWarnings
Formato:

```
public synchronized SQLWarning getWarnings()
```

Devuelve el primer aviso que fue notificado por la última operación SQL que se ejecutó utilizando el contexto de ejecución definido. Los avisos subsiguientes se encadenan al primer aviso. Si no se produce ningún aviso, se devuelve un valor nulo.

getWarnings se utiliza para recuperar SQLCODE positivos.

isBatching
Formato:

```
public synchronized boolean isBatching()
```

Devuelve el valor `true` si se habilita el proceso por lotes en el contexto de ejecución. Devuelve el valor `false` si el proceso por lotes está inhabilitado.

registerStatement
Formato:

```
public RTStatement registerStatement(ConnectionContext connCtx,
 Object profileKey, int stmtNdx)
 throws SQLException
```

Este método lo utiliza código que se genera mediante el conversor SQLJ. No está indicado para el uso directo por parte de los programas de aplicación.

releaseStatement

Formato:

```
public void releaseStatement() throws SQLException
```

Este método lo utiliza código que se genera mediante el conversor SQLJ. No está indicado para el uso directo por parte de los programas de aplicación.

setBatching

Formato:

```
public synchronized void setBatching(boolean proceso_por_lotes)
```

Parámetros:

proceso_por_lotes
Indica si las sentencias que pueden procesarse por lotes y que están registradas con el contexto de ejecución definido pueden añadirse a un lote de sentencias:

true
Las sentencias pueden añadirse a un lote de sentencias.

false
Las sentencias se ejecutan individualmente.

setBatching sólo afecta a las sentencias que se producen en el programa después de invocar a setBatching. No afecta a sentencias anteriores ni a un lote de sentencias existente.

setBatchLimit

Formato:

```
public synchronized void setBatchLimit(int tamaño_del_lote)
```

Establece el número máximo de sentencias que se añaden a un lote antes de ejecutarlo implícitamente.

Parámetros:

tamaño_del_lote
Uno de estos valores:

ExecutionContext.UNLIMITED_BATCH
Indica que la ejecución implícita solo se produce cuando SQLJ encuentra una sentencia que es procesable por lotes pero incompatible, o que no es procesable por lotes. Establecer este valor es lo mismo que no invocar setBatchLimit.

ExecutionContext.AUTO_BATCH
Indica que la ejecución implícita se produce cuando el número de sentencias del lote alcanza un valor definido por SQLJ.

Entero positivo
Es el número de sentencias que se añaden al lote antes de que SQLJ ejecute el lote implícitamente. El lote puede ejecutarse antes de que se

haya añadido este número de sentencias si SQLJ encuentra una sentencia que es procesable por lotes pero incompatible, o que no es procesable por lotes.

setBatchLimit sólo afecta a las sentencias que se producen en el programa después de invocar a setBatchLimit. No afecta a un lote de sentencias existente.

setFetchDirection

Formato:

```
public synchronized void setFetchDirection(int dirección) throws SQLException
```

Proporciona a SQLJ una pista sobre la dirección de captación actual para los objetos del iterador desplazable que se generaron a partir del contexto de ejecución definido.

Parámetros:

dirección
Uno de estos valores:

sqlj.runtime.ResultSetIterator.FETCH_FORWARD
La captación de las filas se realiza hacia adelante. Éste es el valor por omisión.

sqlj.runtime.ResultSetIterator.FETCH_REVERSE
La captación de las filas se realiza hacia atrás.

sqlj.runtime.ResultSetIterator.FETCH_UNKNOWN
El orden de la captación es desconocido.

Cualquier otro valor de entrada da como resultado un SQLException.

setFetchSize

Formato:

```
synchronized public void setFetchSize(int número de filas) throws SQLException
```

Proporciona a SQLJ una pista sobre el número de filas que deben captarse cuando se necesitan más filas.

Parámetros:

número de filas
El número esperado de filas que SQLJ debe captar para el iterador asociado al contexto de ejecución definido.

Si *número de filas* es menor que 0 o mayor que el número máximo de filas que pueden captarse, se emite una excepción de SQL (SQLException).

setMaxFieldSize

Formato:

```
public void setMaxFieldSize(int máximo-bytes)
```

Especifica el número máximo de bytes que se devuelven para cualquier columna (de caracteres, gráfica o binaria de longitud variable) en consultas que utilizan el contexto de ejecución especificado. Si se sobrepasa este límite, SQLJ descarta los bytes restantes.

Parámetros:

máximo_bytes
El número máximo de bytes que SQLJ debe devolver de una columna

BINARY, VARBINARY, CHAR, VARCHAR, GRAPHIC o VARGRAPHIC. Un valor igual a 0 significa que el número de bytes es ilimitado. 0 es el valor por omisión.

setMaxRows

Formato:

```
public synchronized void setMaxRows(int máximo_filas)
```

Especifica el número máximo de filas que son devueltas por una consulta cualquiera que hace uso del contexto de ejecución especificado. Si se sobrepasa este límite, SQLJ descarta las filas restantes.

Cuando se invoca a setMaxRows durante la ejecución en una sentencia SELECT que se ejecuta de forma estática, setMaxRows limita el número máximo de filas de la tabla de resultados sólo mediante el proceso de IBM Data Server Driver para JDBC y SQLJ. La optimización del servidor de datos que limita el número de filas de la tabla de resultados no se produce a menos que también se haya añadido la cláusula FETCH FIRST *n* ROWS ONLY a la sentencia SELECT. Si se ha añadido FETCH FIRST *n* ROWS ONLY a la sentencia SELECT, y se llama a setMaxRows(*m*), el número máximo de filas es el valor más pequeño de *n* y *m*. El controlador descarta el resto de filas.

Parámetros:

máximo_filas
El número máximo de filas que SQLJ debe devolver para una consulta que utiliza el contexto de ejecución definido. Un valor igual a 0 significa que el número de filas es ilimitado. 0 es el valor por omisión.

setQueryTimeout

Formato:

```
public synchronized void setQueryTimeout(int valor_de_tiempo_de_espera)
```

Especifica el número máximo de segundos que pueden ejecutar las operaciones SQL que utilizan el objeto de contexto de ejecución definido. Si una operación SQL sobrepasa el límite, se emite un SQLException.

Para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en servidores de datos DB2 para z/OS, solamente se da soporte a setQueryTimeout si la propiedad queryTimeoutInterruptProcessingMode de Connection o DataSource está establecida en INTERRUPT_PROCESSING_MODE_CLOSE_SOCKET.

Parámetros:

valor_de_tiempo_de_espera
El número máximo de segundos que pueden ejecutar las operaciones SQL que utilizan el objeto de contexto de ejecución definido. 0 significa que el tiempo de ejecución es ilimitado. 0 es el valor por omisión.

Clase sqlj.runtime.SQLNullException

La clase sqlj.runtime.SQLNullException se deriva de la clase java.sql.SQLException.

Se emite una excepción sqlj.runtime.SQLNullException cuando se recupera un valor NULL de SQL en un identificador de sistema principal con un tipo primitivo Java. El valor de SQLSTATE para una instancia de SQLNullException es '22002'.

Clase sqlj.runtime.StreamWrapper

La clase sqlj.runtime.StreamWrapper deriva una instancia de java.io.InputStream y amplía la clase java.io.InputStream.

Las clases sqlj.runtime.AsciiStream, sqlj.runtime.BinaryStream y sqlj.runtime.UnicodeStream amplían sqlj.runtime.StreamWrapper. `sqlj.runtime.StreamWrapper` da soporte a métodos para especificar la longitud de los objetos sqlj.runtime.AsciiStream , sqlj.runtime.BinaryStream y sqlj.runtime.UnicodeStream.

Constructores

StreamWrapper(InputStream)

Formato:

```
protected StreamWrapper(InputStream corriente-entrada)
```

Crea un objeto sqlj.runtime.StreamWrapper con una longitud no especificada.

Parámetros:

corriente-entrada
El objeto InputStream que el objeto sqlj.runtime.StreamWrapper deriva.

StreamWrapper(InputStream, int)

Formato:

```
protected StreamWrapper(java.io.InputStream corriente-entrada, int longitud)
```

Crea un objeto sqlj.runtime.StreamWrapper con una longitud especificada.

Parámetros:

corriente-entrada
El objeto InputStream que el objeto sqlj.runtime.StreamWrapper deriva.

longitud
Longitud del objeto InputStream en bytes.

Métodos

getInputStream

Formato:

```
public InputStream getInputStream()
```

Devuelve el objeto InputStream subyacente derivado mediante el objeto StreamWrapper.

getLength

Formato:

```
public void getLength()
```

Devuelve la longitud en bytes del objeto InputStream derivado, tal como se ha especificado mediante el constructor o en la última llamada a setLength.

setLength

Formato:

```
public void setLength (int longitud)
```

Establece el número de bytes que se leen en el objeto InputStream derivado cuando el objeto se pasa como argumento de entrada a una operación de SQL.

Parámetros:

longitud
Número de bytes que se leen en el objeto InputStream derivado.

Clase sqlj.runtime.UnicodeStream

La clase sqlj.runtime.UnicodeStream se utiliza para una corriente de entrada de datos Unicode con una longitud específica.

La clase sqlj.runtime.UnicodeStream deriva de la clase java.io.InputStream y amplía la clase sqlj.runtime.StreamWrapper. SQLJ interpreta los bytes de un objeto sqlj.runtime.UnicodeStream como caracteres Unicode. Los objetos InputStream con caracteres Unicode deben pasarse como objetos sqlj.runtime.UnicodeStream.

Constructores

UnicodeStream(InputStream)

Formato:

```
public UnicodeStream(java.io.InputStream corriente-entrada)
```

Crea un objeto java.io.InputStream Unicode con una longitud no especificada.

Parámetros:

corriente-entrada
Objeto InputStream que SQLJ interpreta como objeto UnicodeStream .

UnicodeStream(InputStream, int)

Formato:

```
public UnicodeStream(java.io.InputStream corriente-entrada, int longitud)
```

Crea un objeto java.io.InputStream Unicode con una longitud especificada.

Parámetros:

corriente-entrada
Objeto InputStream que SQLJ interpreta como objeto UnicodeStream .

longitud
Longitud del objeto InputStream que SQLJ interpreta como objeto UnicodeStream.

Extensiones JDBC y SQLJ para JDBC en IBM Data Server Driver

IBM Data Server Driver para JDBC y SQLJ proporciona un conjunto de extensiones para el soporte que proporciona la especificación JDBC.

Para utilizar métodos específicos de IBM Data Server Driver para JDBC y SQLJ en clases que tienen las clases estándar correspondientes, convierta cada instancia de la clase JDBC estándar asociada en una instancia de la clase específica de IBM Data Server Driver para JDBC y SQLJ. Por ejemplo:

```
javax.sql.DataSource ds =
 new com.ibm.db2.jcc.DB2SimpleDataSource();
((com.ibm.db2.jcc.DB2BaseDataSource) ds).setServerName("sysmvs1.stl.ibm.com");
```

La Tabla 108 en la página 492 resume las interfaces específicas de IBM Data Server Driver para JDBC y SQLJ.

Tabla 108. Resumen de las interfaces específicas de IBM Data Server Driver para JDBC y SQLJ proporcionadas por IBM Data Server Driver para JDBC y SQLJ

Nombre de interfaz	Fuentes de datos aplicables	Finalidad
DB2CallableStatement	1, 2	Amplía las interfaces java.sql.CallableStatement y com.ibm.db2.jcc.DB2PreparedStatement.
DB2Connection	1, 2, 3	Amplía la interfaz java.sql.Connection.
DB2DatabaseMetaData	1, 2, 3	Amplía la interfaz java.sql.DatabaseMetaData.
DB2Diagnosable	1, 2, 3	Proporciona un mecanismo para obtener los diagnósticos de DB2 a partir de un SQLException de DB2.
DB2ParameterMetaData	2	Amplía la interfaz java.sql.ParameterMetaData.
DB2PreparedStatement	1, 2, 3	Amplía las interfaces com.ibm.db2.jcc.DB2Statement y java.sql.PreparedStatement.
DB2ResultSet	1, 2, 3	Amplía la interfaz java.sql.ResultSet.
DB2RowID	1, 2	Se utiliza para declarar objetos Java para su utilización con el tipo de datos ROWID.
DB2Statement	1, 2, 3	Amplía la interfaz java.sql.Statement.
DB2Struct	2	Proporciona métodos para trabajar con objetos java.sql.Struct.
DB2SystemMonitor	1, 2, 3	Se utiliza para recoger datos de supervisión del sistema para una conexión.
DB2TraceManagerMXBean	1, 2, 3	Proporciona la interfaz MBean para el controlador de rastreo remoto.
DB2Xml	1, 2	Se utiliza para actualizar y recuperar datos de columnas XML.
DBBatchUpdateException	1, 2, 3	Se utiliza para recuperar información de errores sobre la ejecución de sentencias por lotes que devuelven claves generadas automáticamente.

Nota: La interfaz es aplicable a conexiones con las fuentes de datos siguientes:

1. DB2 para z/OS
2. DB2 Database para Linux, UNIX y Windows
3. IBM Informix

La Tabla 109 resume las clases específicas de IBM Data Server Driver para JDBC y SQLJ.

Tabla 109. Resumen de las clases específicas de IBM Data Server Driver para JDBC y SQLJ proporcionadas por IBM Data Server Driver para JDBC y SQLJ

Nombre de clase	Fuentes de datos aplicables	Finalidad
DB2Administrator (solamente para DB2 Database para Linux, UNIX y Windows)	2 en la página 493	Las instancias de la clase DB2Administrator se utilizan para recuperar objetos DB2CataloguedDatabase.
DB2BaseDataSource	1 en la página 493, 2 en la página 493, 3 en la página 493	Clase padre de fuente de datos abstracta para todas las implementaciones de javax.sql.DataSource, javax.sql.ConnectionPoolDataSource y javax.sql.XADataSource específicas de IBM Data Server Driver para JDBC y SQLJ.

Tabla 109. Resumen de las clases específicas de IBM Data Server Driver para JDBC y SQLJ proporcionadas por IBM Data Server Driver para JDBC y SQLJ (continuación)

Nombre de clase	Fuentes de datos aplicables	Finalidad
DB2CataloguedDatabase	2	Contiene métodos para recuperar información sobre una base de datos DB2 Database para Linux, UNIX y Windows.
DB2ClientRerouteServerList	1, 2	Implementa las interfaces java.io.Serializable y javax.naming.Referenceable.
DB2ConnectionPoolDataSource	1, 2, 3	Fábrica de objetos PooledConnection.
DB2DataSource	1, 2, 3	Amplía la clase DB2BaseDataSource e implementa las interfaces javax.sql.DataSource, java.io.Serializable y javax.naming.Referenceable.
DB2Driver	1, 2, 3	Amplía la interfaz java.sql.Driver.
DB2ExceptionFormatter	1, 2, 3	Contiene métodos para la impresión de la información de diagnóstico en una corriente.
DB2JCCPlugin	2	Clase abstracta para la implementación de plugins de seguridad JDBC.
DB2PooledConnection	1, 2, 3	Proporciona métodos que puede utilizar un servidor de aplicaciones para cambiar de usuarios en una conexión fiable preexistente.
DB2PoolMonitor	1, 2	Proporciona métodos para la supervisión de la agrupación de objetos de transporte global para el concentrador de conexiones y el equilibrado de la carga de trabajo Sysplex.
DB2SimpleDataSource	1, 2, 3	Amplía la clase DataBaseDataSource. No soporta agrupaciones de conexiones ni transacciones distribuidas.
DB2Sqlca	1, 2, 3	Encapsulación de la SQLCA de DB2.
DB2TraceManager	1, 2, 3	Controla el grabador de anotaciones cronológicas globales.
DB2Types	1 en la página 492	Define constantes de tipo de datos.
DB2XADataSource	1, 2, 3	Fuente de objetos XADataSource. Un objeto que implementa esta interfaz se registra con un servicio de asignación de nombres que se basa en Java Naming and Directory Interface (JNDI).
DBTimestamp	1, 2, 3	Subclase de Timestamp que gestiona valores de indicación de fecha y hora con precisión adicional o información del huso horario.

Nota: La clase es aplicable a conexiones con las fuentes de datos siguientes:

1. DB2 para z/OS
2. DB2 Database para Linux, UNIX y Windows
3. IBM Informix

Interfaz DBBatchUpdateException

La interfaz com.ibm.db2.jcc.DBBatchUpdateException se utiliza para recuperar información de errores sobre la ejecución por lotes de sentencias que devuelven claves generadas automáticamente.

Métodos de DBBatchUpdateException

Los métodos siguientes se definen únicamente para IBM Data Server Driver para JDBC y SQLJ.

getDBGeneratedKeys

Formato:

```
public java.sql.ResultSet[] getDBGeneratedKeys()
  throws java.sql.SQLException
```

Recupera claves generadas automáticamente que se crearon cuando se ejecutó un lote de sentencias INSERT. Cada objeto ResultSet devuelto contiene las claves generadas automáticamente correspondiente a una sentencia individual del lote. Los objetos ResultSet que son nulos corresponden a sentencias fallidas.

Clase DB2Administrator

Se utilizan instancias de la clase com.ibm.db2.jcc.DB2Administrator para recuperar objetos DB2CataloguedDatabase. La clase DB2Administrator es aplicable solamente a bases de datos DB2 Database para Linux, UNIX y Windows.

Métodos DB2Administrator

getInstance

Formato:

```
public static DB2Administrator getInstance()
```

Devuelve una instancia de la clase DB2Administrator.

getCataloguedDatabases

Formato:

```
public DB2CataloguedDatabase[] getCataloguedDatabases()
 throws java.sql.SQLException
```

Recupera una matriz que contiene un objeto DB2CataloguedDatabase para cada base de datos local del directorio de bases de datos locales.

Si existe un sistema DB2 local y el catálogo no contiene ninguna base de datos, se devuelve una matriz de longitud cero. Si no existe ningún sistema DB2 local, se devuelve un valor nulo. Si el sistema local no es un sistema DB2 Database para Linux, UNIX y Windows, se emite una SQLException.

Clase DB2BaseDataSource

La clase com.ibm.db2.jcc.DB2BaseDataSource es la clase padre de fuente de datos abstracta para todas las implementaciones de javax.sql.DataSource, javax.sql.ConnectionPoolDataSource y javax.sql.XADataSource específicas de IBM Data Server Driver para JDBC y SQLJ.

DB2BaseDataSource implementa la interfaz java.sql.Wrapper.

Propiedades de DB2BaseDataSource

Las propiedades siguientes se definen solamente para IBM Data Server Driver para JDBC y SQLJ.

Puede definir todas las propiedades en un objeto DataSource o en el parámetro *url* de una llamada DriverManager.getConnection.

Todas las propiedades, **excepto** las siguientes, tienen un método setXXX para definir el valor de la propiedad y un método getXXX para recuperar el valor:
- dumpPool
- dumpPoolStatisticsOnSchedule
- dumpPoolStatisticsOnScheduleFile
- maxTransportObjectIdleTime
- maxTransportObjectWaitTime
- minTransportObjects

Un método setXXX tiene este formato:

```
void setNombre-propiedad(tipo-datos valor-propiedad)
```

Un método getXXX tiene este formato:

```
tipo-datos getNombre-propiedad()
```

Nombre-propiedad es el nombre de propiedad no calificado. En el caso de propiedades que no son específicas de IBM Informix, el primer carácter del nombre de la propiedad está en mayúsculas. En el caso de propiedades que únicamente utiliza IBM Informix, todos los caracteres del nombre de la propiedad están en mayúsculas.

La tabla siguiente lista las propiedades de IBM Data Server Driver para JDBC y SQLJ y los tipos de datos asociados a ellas.

Tabla 110. Propiedades de DB2BaseDataSource y tipos de datos asociados a ellas

Nombre de propiedad	Fuentes de datos aplicables	Tipo de datos
com.ibm.db2.jcc.DB2BaseDataSource.accountingInterval	1 en la página 504	String
com.ibm.db2.jcc.DB2BaseDataSource.alternateGroupDatabaseName	1 en la página 504, 2 en la página 504	Serie
com.ibm.db2.jcc.DB2BaseDataSource.alternateGroupPortNumber	1 en la página 504, 2 en la página 504	Serie
com.ibm.db2.jcc.DB2BaseDataSource.alternateGroupServerName	1 en la página 504, 2 en la página 504	Serie
com.ibm.db2.jcc.DB2BaseDataSource.affinityFailbackInterval	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.allowNextOnExhaustedResultSet	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.allowNullResultSetForExecuteQuery	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.atomicMultiRowInsert	1 en la página 504, 2 en la página 504, 3 en la página 504	int

Tabla 110. Propiedades de DB2BaseDataSource y tipos de datos asociados a ellas (continuación)

Nombre de propiedad	Fuentes de datos aplicables	Tipo de datos
com.ibm.db2.jcc.DB2BaseDataSource.blockingReadConnectionTimeout	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.charOutputSize	1 en la página 504	short
com.ibm.db2.jcc.DB2BaseDataSource.clientAccountingInformation	1 en la página 504, 2 en la página 504	String
com.ibm.db2.jcc.DB2BaseDataSource.clientApplicationInformation	1 en la página 504, 2 en la página 504	String
com.ibm.db2.jcc.DB2BaseDataSource.clientDebugInfo (IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4)	1 en la página 504, 2 en la página 504	Serie
com.ibm.db2.jcc.DB2BaseDataSource.clientProgramId	1 en la página 504, 2 en la página 504	Serie
com.ibm.db2.jcc.DB2BaseDataSource.clientProgramName (IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4)	1 en la página 504, 2 en la página 504	Serie
com.ibm.db2.jcc.DB2BaseDataSource.clientRerouteAlternateServerName	1 en la página 504, 2 en la página 504, 3 en la página 504	Serie
com.ibm.db2.jcc.DB2BaseDataSource.clientRerouteAlternatePortNumber	1 en la página 504, 2 en la página 504, 3 en la página 504	Serie
com.ibm.db2.jcc.DB2BaseDataSource.clientRerouteServerListJNDIContext	1 en la página 504, 2 en la página 504, 3 en la página 504	javax.naming.Context
com.ibm.db2.jcc.DB2BaseDataSource.clientRerouteServerListJNDIName	1 en la página 504, 2 en la página 504, 3 en la página 504	Serie
com.ibm.db2.jcc.DB2BaseDataSource.clientUser (IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 sobre DB2 para z/OS solamente)	1 en la página 504	String
com.ibm.db2.jcc.DB2BaseDataSource.clientWorkstation (IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 sobre DB2 para z/OS solamente)	1 en la página 504	String
com.ibm.db2.jcc.DB2BaseDataSource.commandTimeout (IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4)	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.connectionCloseWithInFlightTransaction	1 en la página 504, 2 en la página 504, 3 en la página 504	Serie
com.ibm.db2.jcc.DB2BaseDataSource.concurrentAccessResolution	1 en la página 504, 2 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.connectNode	2 en la página 504	int

Tabla 110. Propiedades de DB2BaseDataSource y tipos de datos asociados a ellas (continuación)

Nombre de propiedad	Fuentes de datos aplicables	Tipo de datos
com.ibm.db2.jcc.DB2BaseDataSource.connectionTimeout (IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4)	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.currentDegree	1 en la página 504, 2 en la página 504	Serie
com.ibm.db2.jcc.DB2BaseDataSource.currentExplainMode	1 en la página 504, 2 en la página 504	Serie
com.ibm.db2.jcc.DB2BaseDataSource.currentExplainSnapshot	2 en la página 504	Serie
com.ibm.db2.jcc.DB2BaseDataSource.currentFunctionPath	1 en la página 504, 2 en la página 504	String
currentLocaleLcCtype	1 en la página 504	Serie
com.ibm.db2.jcc.DB2BaseDataSource.currentLockTimeout	2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.currentMaintainedTableTypesForOptimization	1 en la página 504, 2 en la página 504	Serie
com.ibm.db2.jcc.DB2BaseDataSource.currentPackagePath	1 en la página 504, 2 en la página 504	String
com.ibm.db2.jcc.DB2BaseDataSource.currentPackageSet	1 en la página 504, 2 en la página 504	String
com.ibm.db2.jcc.DB2BaseDataSource.currentQueryOptimization	2 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.currentRefreshAge	1 en la página 504, 2 en la página 504	long
com.ibm.db2.jcc.DB2BaseDataSource.currentSchema	1 en la página 504, 2 en la página 504	String
com.ibm.db2.jcc.DB2BaseDataSource.cursorSensitivity	1 en la página 504, 2 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.currentSQLID	1 en la página 504	String
com.ibm.db2.jcc.DB2BaseDataSource.databaseName	1 en la página 504, 2 en la página 504, 3 en la página 504	String
com.ibm.db2.jcc.DB2BaseDataSource.dateFormat	1 en la página 504, 2 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.decimalRoundingMode	1 en la página 504, 2 en la página 504	int

Tabla 110. Propiedades de DB2BaseDataSource y tipos de datos asociados a ellas (continuación)

Nombre de propiedad	Fuentes de datos aplicables	Tipo de datos
com.ibm.db2.jcc.DB2BaseDataSource.decimalSeparator	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.decimalStringFormat	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.defaultIsolationLevel	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.deferPrepares	1 en la página 504, 2 en la página 504, 3 en la página 504	boolean
com.ibm.db2.jcc.DB2BaseDataSource.description	1 en la página 504, 2 en la página 504, 3 en la página 504	String
com.ibm.db2.jcc.DB2BaseDataSource.downgradeHoldCursorsUnderXa	1 en la página 504, 2 en la página 504,3 en la página 504	boolean
com.ibm.db2.jcc.DB2BaseDataSource.driverType	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.dumpPool	3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.dumpPoolStatisticsOnSchedule	3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.dumpPoolStatisticsOnScheduleFile	3 en la página 504	Serie
com.ibm.db2.jcc.DB2BaseDataSource.enableAlternateGroupSeamlessACR	1 en la página 504, 2 en la página 504	boolean
com.ibm.db2.jcc.DB2BaseDataSource.enableClientAffinitiesList	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.enableExtendedIndicators	1 en la página 504, 2 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.enableNamedParameterMarkers	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.enableConnectionConcentrator (IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4)	1 en la página 504, 3 en la página 504	boolean
com.ibm.db2.jcc.DB2BaseDataSource.enableMultiRowInsertSupport	1 en la página 504	boolean
com.ibm.db2.jcc.DB2BaseDataSource.enableRowsetSupport	1 en la página 504, 2 en la página 504	int

Tabla 110. Propiedades de DB2BaseDataSource y tipos de datos asociados a ellas (continuación)

Nombre de propiedad	Fuentes de datos aplicables	Tipo de datos
com.ibm.db2.jcc.DB2BaseDataSource.enableSeamlessFailover	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.enableSysplexWLB (IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4)	1 en la página 504, 2 en la página 504, 3 en la página 504	boolean
com.ibm.db2.jcc.DB2BaseDataSource.encryptionAlgorithm	1 en la página 504, 2 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.enableExtendedDescribe	1 en la página 504, 2 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.fetchSize	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.floatingPointStringFormat	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.fullyMaterializeInputStreams	1 en la página 504, 2 en la página 504	boolean
com.ibm.db2.jcc.DB2BaseDataSource.fullyMaterializeLobData	1 en la página 504, 2 en la página 504, 3 en la página 504	boolean
com.ibm.db2.jcc.DB2BaseDataSource.gssCredential	1 en la página 504, 2 en la página 504	Object
com.ibm.db2.jcc.DB2BaseDataSource.implicitRollbackOption	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.interruptProcessingMode (IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4)	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.jdbcCollection	1 en la página 504	Serie
com.ibm.db2.jcc.DB2BaseDataSource.keepAliveTimeout (IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4)	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.keepDynamic	1 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.kerberosServerPrincipal	1 en la página 504, 2 en la página 504	String
com.ibm.db2.jcc.DB2BaseDataSource.loginTimeout (no válido para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 sobre DB2 para z/OS)	1 en la página 504, 2 en la página 504, 3 en la página 504	int

Tabla 110. Propiedades de DB2BaseDataSource y tipos de datos asociados a ellas (continuación)

Nombre de propiedad	Fuentes de datos aplicables	Tipo de datos
com.ibm.db2.jcc.DB2BaseDataSource.logWriter	1 en la página 504, 2 en la página 504, 3 en la página 504	PrintWriter
com.ibm.db2.jcc.DB2BaseDataSource.maxConnCachedParamBufferSize (IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 sólo en DB2 para z/OS)	1 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.maxRetriesForClientReroute	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.maxStatements	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.maxRowsetSize (IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS solamente)	1 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.maxTransportObjectIdleTime	3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.maxTransportObjectWaitTime	3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.maxTransportObjects	1 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.memberConnectTimeout (IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4)	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.minTransportObjects	3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.optimizationProfile	2 en la página 504	Serie
com.ibm.db2.jcc.DB2BaseDataSource.optimizationProfileToFlush	2 en la página 504	Serie
com.ibm.db2.jcc.DB2BaseDataSource.password	1 en la página 504, 2 en la página 504, 3 en la página 504	Serie
com.ibm.db2.jcc.DB2BaseDataSource.pdqProperties	1 en la página 504, 2 en la página 504	Serie
com.ibm.db2.jcc.DB2BaseDataSource.pkList (IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2)	1 en la página 504	Serie
com.ibm.db2.jcc.DB2BaseDataSource.planName (solamente para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2)	1 en la página 504	Serie
com.ibm.db2.jcc.DB2BaseDataSource.plugin	2 en la página 504	Object
com.ibm.db2.jcc.DB2BaseDataSource.pluginName	2 en la página 504	Serie
com.ibm.db2.jcc.DB2BaseDataSource.portNumber	1 en la página 504, 2 en la página 504, 3 en la página 504	int

Tabla 110. Propiedades de DB2BaseDataSource y tipos de datos asociados a ellas (continuación)

Nombre de propiedad	Fuentes de datos aplicables	Tipo de datos
com.ibm.db2.jcc.DB2BaseDataSource.progressiveStreaming	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.queryCloseImplicit	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.queryDataSize	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.queryTimeoutInterruptProcessingMode	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.readOnly	1 en la página 504, 2 en la página 504	boolean
com.ibm.db2.jcc.DB2BaseDataSource.reportLongTypes	1 en la página 504	short
com.ibm.db2.jcc.DB2BaseDataSource.resultSetHoldability	1 en la página 504, 2 en la página 504,3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.resultSetHoldabilityForCatalogQueries	1 en la página 504, 2 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.retrieveMessagesFromServerOnGetMessage	1 en la página 504, 2 en la página 504, 3 en la página 504	boolean
com.ibm.db2.jcc.DB2BaseDataSource.retryIntervalForClientReroute	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.retryWithAlternativeSecurityMechanism (IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4)	2 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.returnAlias	1 en la página 504, 2 en la página 504	short
com.ibm.db2.jcc.DB2BaseDataSource.securityMechanism	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.sendCharInputsUTF8	1 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.sendDataAsIs	1 en la página 504, 2 en la página 504, 3 en la página 504	boolean
com.ibm.db2.jcc.DB2BaseDataSource.serverName	1 en la página 504, 2 en la página 504, 3 en la página 504	Serie

Tabla 110. Propiedades de DB2BaseDataSource y tipos de datos asociados a ellas (continuación)

Nombre de propiedad	Fuentes de datos aplicables	Tipo de datos
com.ibm.db2.jcc.DB2BaseDataSource.sessionTimeZone	1 en la página 504	Serie
com.ibm.db2.jcc.DB2BaseDataSource.sqljEnableClassLoaderSpecificProfiles	1 en la página 504	boolean
com.ibm.db2.jcc.DB2BaseDataSource.ssid (IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 sobre DB2 para z/OS solamente)	1 en la página 504	Serie
com.ibm.db2.jcc.DB2BaseDataSource.sslConnection (IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4)	1 en la página 504, 2 en la página 504, 3 en la página 504	boolean
com.ibm.db2.jcc.DB2BaseDataSource.sslTrustStoreLocation (IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4)	1 en la página 504, 2 en la página 504, 3 en la página 504	Serie
com.ibm.db2.jcc.DB2BaseDataSource.sslTrustStorePassword (IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4)	1 en la página 504, 2 en la página 504, 3 en la página 504	Serie
com.ibm.db2.jcc.DB2BaseDataSource.statementConcentrator	1 en la página 504, 2 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.streamBufferSize	1 en la página 504, 2 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.stripTrailingZerosForDecimalNumbers	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.supportsAsynchronousXARollback	1 en la página 504, 2 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.sysSchema	1 en la página 504, 2 en la página 504	Serie
com.ibm.db2.jcc.DB2BaseDataSource.timerLevelForQueryTimeOut	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.timeFormat	1 en la página 504, 2 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.timestampFormat	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.timestampOutputType	1 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.timestampPrecisionReporting	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.traceDirectory	1 en la página 504, 2 en la página 504, 3 en la página 504	Serie

Tabla 110. Propiedades de DB2BaseDataSource y tipos de datos asociados a ellas (continuación)

Nombre de propiedad	Fuentes de datos aplicables	Tipo de datos
com.ibm.db2.jcc.DB2BaseDataSource.traceFile	1 en la página 504, 2 en la página 504, 3 en la página 504	Serie
com.ibm.db2.jcc.DB2BaseDataSource.traceFileAppend	1 en la página 504, 2 en la página 504, 3 en la página 504	boolean
com.ibm.db2.jcc.DB2BaseDataSource.traceFileCount	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.traceFileSize	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.traceLevel	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.traceOption	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.useCachedCursor	1 en la página 504, 2 en la página 504	boolean
com.ibm.db2.jcc.DB2BaseDataSource.useJDBC4ColumnNameAndLabelSemantics	1 en la página 504, 2 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.useJDBC41DefinitionForGetColumns	1 en la página 504, 2 en la página 504, 3 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.user	1 en la página 504, 2 en la página 504, 3 en la página 504	Serie
com.ibm.db2.jcc.DB2BaseDataSource.useIdentityValLocalForAutoGeneratedKeys	1 en la página 504, 2 en la página 504	boolean
com.ibm.db2.jcc.DB2BaseDataSource.useRowsetCursor	1 en la página 504	boolean
com.ibm.db2.jcc.DB2BaseDataSource.useTransactionRedirect	2 en la página 504	boolean
com.ibm.db2.jcc.DB2BaseDataSource.xaNetworkOptimization	1 en la página 504, 2 en la página 504, 3 en la página 504	boolean
com.ibm.db2.jcc.DB2BaseDataSource.xmlFormat	1 en la página 504, 2 en la página 504	int
com.ibm.db2.jcc.DB2BaseDataSource.DBANSIWARN	3 en la página 504	boolean
com.ibm.db2.jcc.DB2BaseDataSource.DBDATE	3 en la página 504	Serie

Tabla 110. Propiedades de DB2BaseDataSource y tipos de datos asociados a ellas (continuación)

Nombre de propiedad	Fuentes de datos aplicables	Tipo de datos
com.ibm.db2.jcc.DB2BaseDataSource.DBPATH	3	Serie
com.ibm.db2.jcc.DB2BaseDataSource.DBSPACETEMP	3	Serie
com.ibm.db2.jcc.DB2BaseDataSource.DBTEMP	3	Serie
com.ibm.db2.jcc.DB2BaseDataSource.DBUPSPACE	3	Serie
com.ibm.db2.jcc.DB2BaseDataSource.DELIMIDENT	3	boolean
com.ibm.db2.jcc.DB2BaseDataSource.IFX_DIRECTIVES	3	Serie
com.ibm.db2.jcc.DB2BaseDataSource.IFX_EXTDIRECTIVES	3	Serie
com.ibm.db2.jcc.DB2BaseDataSource.IFX_UPDDESC	3	Serie
com.ibm.db2.jcc.DB2BaseDataSource.IFX_XASTDCOMPLIANCE_XAEND	3	Serie
com.ibm.db2.jcc.DB2BaseDataSource.INFORMIXOPCACHE	3	Serie
com.ibm.db2.jcc.DB2BaseDataSource.INFORMIXSTACKSIZE	3	Serie
com.ibm.db2.jcc.DB2BaseDataSource.NODEFDAC	3	Serie
com.ibm.db2.jcc.DB2BaseDataSource.OPTCOMPIND	3	Serie
com.ibm.db2.jcc.DB2BaseDataSource.OPTOFC	3	Serie
com.ibm.db2.jcc.DB2BaseDataSource.PDQPRIORITY	3	Serie
com.ibm.db2.jcc.DB2BaseDataSource.PSORT_DBTEMP	3	Serie
com.ibm.db2.jcc.DB2BaseDataSource.PSORT_NPROCS	3	Serie
com.ibm.db2.jcc.DB2BaseDataSource.STMT_CACHE	3	Serie

Nota: La propiedad es aplicable a conexiones con las fuentes de datos siguientes:

1. DB2 para z/OS
2. DB2 Database para Linux, UNIX y Windows
3. IBM Informix

Campos de DB2BaseDataSource

Las constantes siguientes se definen solamente para IBM Data Server Driver para JDBC y SQLJ.

public final static int IMPLICIT_ROLLBACK_OPTION_NOT_SET = 0
Constante para la propiedad implicitRollbackOption. Este valor indica que una conexión no se cierra cuando se produce un punto muerto o un tiempo de espera excedido. Este valor genera el mismo comportamiento que IMPLICIT_ROLLBACK_OPTION_NOT_CLOSE_CONNECTION.

public final static int IMPLICIT_ROLLBACK_OPTION_NOT_CLOSE_CONNECTION = 1
Constante para la propiedad implicitRollbackOption. Este valor indica que una conexión no se cierra cuando se produce un punto muerto o un tiempo de espera excedido. IBM Data Server Driver para JDBC y SQLJ devuelve el código de error que el servidor de datos genera para un punto muerto o un tiempo de espera excedido.

public final static int IMPLICIT_ROLLBACK_OPTION_CLOSE_CONNECTION = 2
Constante para la propiedad implicitRollbackOption. Este valor indica que una conexión se cierra cuando se produce un punto muerto o un tiempo de espera excedido.

public final static int INTERRUPT_PROCESSING_MODE_DISABLED = 0
Constante para la propiedad interruptProcessingMode. Este valor indica que está inhabilitado el proceso de interrupciones.

public final static int INTERRUPT_PROCESSING_MODE_STATEMENT_CANCEL = 1
Constante para la propiedad interruptProcessingMode. Este valor indica que IBM Data Server Driver para JDBC y SQLJ cancela la sentencia en ejecución actualmente cuando una aplicación ejecuta Statement.cancel, si el servidor de datos da soporte al proceso de interrupciones.

public final static int INTERRUPT_PROCESSING_MODE_CLOSE_SOCKET = 2
Constante para la propiedad interruptProcessingMode. Este valor indica que IBM Data Server Driver para JDBC y SQLJ descarta el socket subyacente y cierra la conexión cuando una aplicación ejecuta Statement.cancel.

public final static int NOT_SET = 0
Valor por omisión para las propiedades.

public final static int YES = 1
Valor YES para las propiedades.

public final static int NO = 2
Valor NO para las propiedades.

public final static int QUERYTIMEOUT_DISABLED = -1
Constante para la propiedad timerLevelForQueryTimeOut. Este valor indica que no se crearán objetos Timer para esperar a que las consultas excedan el tiempo de espera.

public final static int QUERYTIMEOUT_STATEMENT_LEVEL = 1
Constante para la propiedad timerLevelForQueryTimeOut. Este valor indica que se crearán objetos Timer en el nivel Statement para esperar a que las consultas excedan el tiempo de espera.

public final static int QUERYTIMEOUT_CONNECTION_LEVEL = 2
Constante para la propiedad timerLevelForQueryTimeOut. Este valor indica que se crearán objetos Timer en el nivel Connection para esperar a que las consultas excedan el tiempo de espera.

public final static int TRACE_OPTION_CIRCULAR = 1
Constante para la propiedad traceOption. Este valor indica que IBM Data Server Driver para JDBC y SQLJ utiliza el rastreo circular.

Métodos de DB2BaseDataSource

Además de los métodos getXXX y setXXX para las propiedades de DB2BaseDataSource, están definidos los métodos siguientes solo para IBM Data Server Driver para JDBC y SQLJ.

getReference
Formato:

```
public javax.naming.Reference getReference()
  throws javax.naming.NamingException
```

Obtiene la referencia (Reference) de un objeto DataSource. Para obtener una explicación sobre Reference, consulte la descripción de javax.naming.Referenceable en la documentación de Java Platform Standard Edition.

Interfaz DB2CallableStatement

La interfaz com.ibm.db2.jcc.DB2CallableStatement amplía las interfaces java.sql.CallableStatement y com.ibm.db2.jcc.DB2PreparedStatement.

Métodos DB2CallableStatement

Los métodos siguientes se definen únicamente para IBM Data Server Driver para JDBC y SQLJ.

getDBTimestamp

Formatos:

```
public DBTimestamp getDBTimestamp(int
Índice_Parámetro)
  throws SQLException
public DBTimestamp getDBTimestamp(String
NombreParámetro)
  throws SQLException
```

Devuelve el valor de un parámetro TIMESTAMP OUT o INOUT como objeto DBTimestamp. Si el valor del parámetro es NULL, el valor que se devuelve es nulo.

Parámetros:

índiceParámetro
Número del parámetro cuyo valor se está recuperando.

nombreParámetro
Nombre del parámetro cuyo valor se está recuperando.

No se admite este método para las conexiones con IBM Informix.

getJccArrayAtName

Formato:

```
public java.sql.Array getJccArrayAtName(String
nombreMarcadorParámetro)
  throws java.sql.SQLException
```

Recupera un valor ARRAY designado por un marcador de parámetro con nombre como valor java.sql.Array.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en `DB2BaseDataSource.YES` (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro para el que se recupera un valor.

getJccBigDecimalAtName

Formato:

```
public java.math.BigDecimal
getJccBigDecimalAtName(String
nombreMarcadorParámetro)
  throws java.sql.SQLException
public java.math.BigDecimal getJccBigDecimalAtName(String
nombreMarcadorParámetro,
  int escala)
  throws java.sql.SQLException
```

Recupera un valor DECIMAL designado por un marcador de parámetro con nombre como valor java.math.BigDecimal.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en `DB2BaseDataSource.YES` (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro para el que se recupera un valor.

escala
Escala del valor que se recupera.

getJccBlobAtName

Formatos:

```
public java.sql.Blob getJccBlobAtName(String
nombreMarcadorParámetro)
  throws java.sql.SQLException
```

Recupera un valor BLOB designado por un marcador de parámetro con nombre como valor java.sql.Blob.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en `DB2BaseDataSource.YES` (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro para el que se recupera un valor.

getJccBooleanAtName

Formato:

```
public boolean getJccBooleanAtName(String
nombreMarcadorParámetro)
  throws java.sql.SQLException
```

Recupera un valor BIT o BOOLEAN designado por un marcador de parámetro con nombre como valor booleano.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en `DB2BaseDataSource.YES` (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro para el que se recupera un valor.

getJccByteAtName

Formato:

```
public byte getJccByteAtName(String
nombreMarcadorParámetro)
  throws java.sql.SQLException
```

Recupera un valor TINYINT designado por un marcador de parámetro con nombre como valor de byte.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en `DB2BaseDataSource.YES` (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro para el que se recupera un valor.

getJccBytesAtName

Formato:

```
public byte[] getJccBytesAtName(String
nombreMarcadorParámetro)
  throws java.sql.SQLException
```

Recupera un valor BINARY o VARBINARY designado por un marcador de parámetro con nombre como matriz de valores de byte.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en DB2BaseDataSource.YES (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro para el que se recupera un valor.

getJccClobAtName

Formato:

```
public java.sql.Blob getJccClobAtName(String
nombreMarcadorParámetro)
  throws java.sql.SQLException
```

Recupera un valor CLOB designado por un marcador de parámetro con nombre como valor java.sql.Clob.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en DB2BaseDataSource.YES (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro para el que se recupera un valor.

getJccDateAtName

Formatos:

```
public java.sql.Date getJccDateAtName(String
nombreMarcadorParámetro)
  throws java.sql.SQLException
public java.sql.Date getJccDateAtName(String
nombreMarcadorParámetro,
  java.util.Calendar cal)
  throws java.sql.SQLException
```

Recupera un valor DATE designado por un marcador de parámetro con nombre como valor java.sql.Date.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en DB2BaseDataSource.YES (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro para el que se recupera un valor.

cal
El objeto java.util.Calendar que utiliza IBM Data Server Driver para JDBC y SQLJ para construir la fecha.

getJccDoubleAtName

Formato:

```
public double getJccDoubleAtName(String
nombreMarcadorParámetro)
  throws java.sql.SQLException
```

Recupera un valor DOUBLE designado por un marcador de parámetro con nombre como valor double.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en DB2BaseDataSource.YES (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro para el que se recupera un valor.

getJccFloatAtName

Formato:

```
public double getJccFloatAtName(String
nombreMarcadorParámetro)
  throws java.sql.SQLException
```

Recupera un valor FLOAT designado por un marcador de parámetro con nombre como valor double.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en `DB2BaseDataSource.YES` (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro para el que se recupera un valor.

getJccIntAtName

Formato:

```
public int getJccIntAtName(String
nombreMarcadorParámetro)
  throws java.sql.SQLException
```

Recupera un valor INTEGER designado por un marcador de parámetro con nombre como valor int.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en `DB2BaseDataSource.YES` (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro para el que se recupera un valor.

getJccLongAtName

Formato:

```
public long getJccLongAtName(String
nombreMarcadorParámetro)
  throws java.sql.SQLException
```

Recupera un valor BIGINT designado por un marcador de parámetro con nombre como valor long.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en `DB2BaseDataSource.YES` (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro para el que se recupera un valor.

getJccObjectAtName

Formatos:

```
public java.sql.Object getJccObjectAtName(String
nombreMarcadorParámetro)
  throws java.sql.SQLException
```

```
public java.sql.Object getJccObjectAtName(String
nombreMarcadorParámetro,
  Map correlación)
  throws java.sql.SQLException
```

Recupera un valor designado por un marcador de parámetro con nombre como valor java.sql.Object.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en DB2BaseDataSource.YES (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro para el que se recupera un valor.

correlación
La correlación de nombres de tipos SQL con clases de Java.

getJccRowIdAtName

Formato:

```
public java.sql.RowId getJccRowIdAtName(String
nombreMarcadorParámetro)
  throws java.sql.SQLException
```

Recupera un valor ROWID designado por un marcador de parámetro con nombre como valor java.sql.RowId.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en DB2BaseDataSource.YES (1).

Este método requiere IBM Data Server Driver para JDBC y SQLJ Versión 4.8 o posterior.

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro para el que se recupera un valor.

getJccShortAtName

Formato:

```
public short getJccShortAtName(String
nombreMarcadorParámetro)
  throws java.sql.SQLException
```

Recupera un valor SMALLINT designado por un marcador de parámetro con nombre como valor short.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en DB2BaseDataSource.YES (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro para el que se recupera un valor.

getJccSQLXMLAtName

Formato:

```
public java.sql.SQLXML getJccSQLXMLAtName(String
nombreMarcadorParámetro)
  throws java.sql.SQLException
```

Recupera un valor SQLXML designado por un marcador de parámetro con nombre como valor java.sql.SQLXML.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en DB2BaseDataSource.YES (1).

Este método requiere IBM Data Server Driver para JDBC y SQLJ Versión 4.8 o posterior.

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro para el que se recupera un valor.

getJccStringAtName

Formato:

```
public java.lang.String getJccStringAtName(String
nombreMarcadorParámetro)
  throws java.sql.SQLException
```

Recupera un valor CHAR, VARcHAR o LONGVARCHAR designado por un marcador de parámetro con nombre como valor java.lang.String.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en DB2BaseDataSource.YES (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro para el que se recupera un valor.

getJccTimeAtName

Formatos:

```
public java.sql.Time getJccTimeAtName(String
nombreMarcadorParámetro)
  throws java.sql.SQLException
public java.sql.Time getJccTimeAtName(String
nombreMarcadorParámetro,
  java.util.Calendar cal)
  throws java.sql.SQLException
```

Recupera un valor TIME designado por un marcador de parámetro con nombre como valor java.sql.Time.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en DB2BaseDataSource.YES (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro para el que se recupera un valor.

cal
El objeto java.util.Calendar que utiliza IBM Data Server Driver para JDBC y SQLJ para construir la hora.

getJccTimestampAtName

Formatos:

```
public java.sql.Timestamp
getJccTimestampAtName(String
nombreMarcadorParámetro)
  throws java.sql.SQLException
public java.sql.Timestamp getJccTimestampAtName(String
nombreMarcadorParámetro,
  java.util.Calendar cal)
  throws java.sql.SQLException
```

Recupera un valor TIMESTAMP designado por un marcador de parámetro con nombre como valor java.sql.Timestamp.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en DB2BaseDataSource.YES (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro para el que se recupera un valor.

cal
El objeto java.util.Calendar que utiliza IBM Data Server Driver para JDBC y SQLJ para construir la indicación de fecha y hora.

registerJccOutParameterAtName

Formatos:

```
public void registerJccOutParameterAtName(String nombreMarcadorParámetro,
  int tiposql)
  throws java.sql.SQLException
public void registerJccOutParameterAtName(String nombreMarcadorParámetro,
  int tiposql,
  int escala)
  throws java.sql.SQLException
public void registerJccOutParameterAtName(String nombreMarcadorParámetro,
  int tiposql,
  String typeName)
  throws java.sql.SQLException
```

Registra un parámetro OUT que *nombreMarcadorParámetro* identifica como el *tiposql* de tipo JDBC.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en DB2BaseDataSource.YES (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro para el parámetro que se va a registrar.

tiposql
Código de tipo JDBC, tal y como se define en java.sql.Types, del parámetro que se va a registrar.

escala
Escala del parámetro que se va a registrar. Este parámetro se aplica únicamente al caso siguiente:

- Si *tiposql* es java.sql.Types.DECIMAL o java.sql.Types.NUMERIC, *escala* es el número de dígitos situados a la derecha de la coma decimal.

typeName
Si *Tipojdbc* es java.sql.Types.DISTINCT o java.sql.Types.REF, el nombre totalmente calificado del tipo SQL definido por el usuario del parámetro que se va a registrar.

setDBTimestamp

Formato:

```
public void setDBTimestamp(String
nombreParámetro,
  DBTimestamp IndicaciónFechaHora)
  throws java.sql.SQLException
```

Asigna un valor DBTimestamp como parámetro IN o INOUT.

Parámetros:

nombreParámetro
Nombre del parámetro al que se asigna un valor de variable DBTimestamp.

IndicaciónFechaHora
Valor DBTimestamp que se asigna al marcador de parámetro.

No se admite este método para las conexiones con IBM Informix.

Métodos setJccXXXAtName
Estos métodos proceden de DB2PreparedStatement.

Clase DB2CataloguedDatabase

La clase com.ibm.db2.jcc.DB2CataloguedDatabase contiene métodos que recuperan información sobre una base de datos local DB2 Database para Linux, UNIX y Windows.

No es necesaria ninguna conexión de base de datos para invocar métodos DB2CataloguedDatabase.

Métodos DB2CataloguedDatabase

getServerName
Formato:

```
public String getServerName()
```

Recupera el nombre del servidor donde reside la base de datos.

getPortNumber
Formato:

```
public int getPortNumber()
```

Recupera el número de puerto asociado a la instancia de DB2.

getDatabaseName
Formato:

```
public String getDatabaseName()
```

Recupera el nombre de la base de datos.

getDatabaseAlias
Formato:

```
public String getDatabaseAlias()
```

Recupera el alias de la base de datos.

Clase DB2ClientRerouteServerList

La clase com.ibm.db2.jcc.DB2ClientRerouteServerList implementa las interfaces java.io.Serializable y javax.naming.Referenceable.

Métodos DB2ClientRerouteServerList

getAlternatePortNumber
Formato:

```
public int[] getAlternatePortNumber()
```

Recupera los números de puerto correspondientes a los servidores alternativos.

getAlternateServerName
Formato:

```
public String[] getAlternateServerName()
```

Recupera una matriz que contiene los nombres de los servidores alternativos. Estos valores son direcciones IP o nombres de servidor DNS.

getPrimaryPortNumber
Formato:

```
public int getPrimaryPortNumber()
```

Recupera el número de puerto asociado al servidor primario.

getPrimaryServerName
Formato:

```
public String[] getPrimaryServerName()
```

Recupera el nombre del servidor primario. Este valor es una dirección IP o un nombre de servidor DNS.

setAlternatePortNumber
Formato:

```
public void setAlternatePortNumber(int[] listaNúmeroPuertoAlternativo)
```

Establece los números de puerto correspondientes a los servidores alternativos.

setAlternateServerName
Formato:

```
public void setAlternateServerName(String[] servidor-alternativo)
```

Define los nombres de servidores alternativos. Estos valores son direcciones IP o nombres de servidor DNS.

setPrimaryPortNumber
Formato:

```
public void setPrimaryPortNumber(int númeroPuertoPrimario)
```

Define el número de puerto asociado al servidor primario.

setPrimaryServerName
Formato:

```
public void setPrimaryServerName(String servidor-primario)
```

Define el nombre del servidor primario. Este valor es una dirección IP o un nombre de servidor DNS.

Interfaz DB2Connection

La interfaz com.ibm.db2.jcc.DB2Connection amplía la interfaz java.sql.Connection.

DB2Connection implementa la interfaz java.sql.Wrapper.

Métodos de DB2Connection

Los métodos siguientes se definen únicamente para IBM Data Server Driver para JDBC y SQLJ.

alternateWasUsedOnConnect
Formato:

```
public boolean alternateWasUsedOnConnect()
  throws java.sql.SQLException
```

Devuelve `true` si el controlador utilizó información sobre el servidor alternativo para obtener la conexión. La información sobre el servidor alternativo está disponible en la información transitoria de clientRerouteServerList en DB2BaseDataSource, que el servidor de bases de datos actualiza cuando cambian los servidores primario y alternativo.

changeDB2Password

Formato:

```
public abstract void changeDB2Password(String oldPassword,
  String newPassword)
  throws java.sql.SQLException
```

Cambia la contraseña para acceder a la fuente de datos, para el usuario del objeto Connection.

Descripciones de parámetros:

oldPassword
: Contraseña original de Connection.

newPassword
: Contraseña nueva de Connection.

createArrayOf

Formato:

```
Array createArrayOf(String typeName,
  Object[] elements)
  throws SQLException;
```

Crea un objeto java.sql.Array.

Descripciones de parámetros:

typeName
: Tipo de datos de SQL de los elementos de la matriz. *typeName* puede ser un tipo de datos incorporado o un tipo diferenciado.

elements
: Elementos utilizados para llenar el objeto Array.

createStruct

Formato:

```
Struct createStruct(String nombre-tipo,
  Object[] atributos)
  throws SQLException;
```

Devuelve un objeto java.sql.Struct que se correlaciona con *nombre-tipo* y que tiene los atributos especificados en *atributos*.

Descripciones de parámetros:

typeName
: Tipo de datos SQL del tipo estructurado SQL con el que se correlaciona el objeto *Struct*. *nombre-tipo* es el nombre de un tipo definido por el usuario que se ha establecido en el servidor de datos.

atributos
: Atributos utilizados para llenar el objeto Struct devuelto.

deregisterDB2XmlObject

Formatos:

```
public void deregisterDB2XmlObject(String sqlIdSchema,
  String sqlIdName)
  throws SQLException
```

Elimina un esquema XML registrado previamente de la fuente de datos.

Descripciones de parámetros:

sqlIdSchema
Nombre del esquema SQL para el esquema XML. *sqlIdSchema* es un valor de serie con una longitud máxima de 128 bytes. El valor de *sqlIdSchema* debe seguir las reglas de denominación aplicables a cualquier nombre de esquema de SQL. El nombre no puede empezar por la serie 'SYS'. Si el valor de *sqlIdSchema* es nulo, el sistema de bases de datos utilizará el valor del registro especial CURRENT SCHEMA.

sqlIdName
Nombre SQL del esquema XML. *sqlIdName* es un valor de serie con una longitud máxima de 128 bytes. El valor de *sqlIdName* debe seguir las reglas aplicables a un identificador de SQL. Si el valor de *sqlIdSchema* es nulo, el valor de *sqlIdName* puede ser nulo. En este caso, el sistema de bases de datos genera el valor de *sqlIdName*.

getDB2ClientAccountingInformation

Formato:

```
public String getDB2ClientAccountingInformation()
  throws SQLException
```

Devuelve información de contabilidad para el cliente actual.

Importante: getDB2ClientAccountingInformation está en desuso en la implementación para JDBC 4.0 de IBM Data Server Driver para JDBC y SQLJ. En su lugar utilice java.sql.Connection.getClientInfo.

getDB2ClientApplicationInformation

Formato:

```
public String getDB2ClientApplicationInformation()
  throws java.sql.SQLException
```

Devuelve información de aplicación para el cliente actual.

Importante: getDB2ClientApplicationInformation está en desuso en la implementación para JDBC 4.0 de IBM Data Server Driver para JDBC y SQLJ. En su lugar utilice java.sql.Connection.getClientInfo.

getDB2ClientProgramId

Formato:

```
public String getDB2ClientProgramId()
  throws java.sql.SQLException
```

Devuelve el identificador del programa definido por el usuario para el cliente. El identificador de programa se puede utilizar para identificar la aplicación en la fuente de datos.

getDB2ClientProgramId no se aplica a los servidores de datos de DB2 Database para Linux, UNIX y Windows.

getDB2ClientUser

Formato:

```
public String getDB2ClientUser()
  throws java.sql.SQLException
```

Devuelve el nombre de usuario del cliente actual para la conexión. Este nombre no es el valor de usuario para la conexión JDBC.

Importante: getDB2ClientUser está en desuso en la implementación para JDBC 4.0 de IBM Data Server Driver para JDBC y SQLJ. En su lugar utilice java.sql.Connection.getClientInfo.

getDB2ClientWorkstation

Formato:

```
public String getDB2ClientWorkstation()
  throws java.sql.SQLException
```

Devuelve el nombre de estación de trabajo para el cliente actual.

Importante: getDB2ClientWorkstation está en desuso en la implementación para JDBC 4.0 de IBM Data Server Driver para JDBC y SQLJ. En su lugar utilice java.sql.Connection.getClientInfo.

getDB2Correlator

Formato:

```
String getDB2Correlator()
  throws java.sql.SQLException
```

Devuelve el valor de la variable de instancia (símbolo de correlación) crrtkn que DRDA envía con el mandato ACCRDB. El símbolo de correlación únicamente identifica una conexión lógica con un servidor.

getDB2CurrentPackagePath

Formato:

```
public String getDB2CurrentPackagePath()
  throws java.sql.SQLException
```

Devuelve una lista de colecciones de paquetes de DB2 en los que se buscan los paquetes JDBC y SQLJ.

El método getDB2CurrentPackagePath solamente se aplica a conexiones con sistemas de bases de datos de DB2.

getDB2CurrentPackageSet

Formato:

```
public String getDB2CurrentPackageSet()
  throws java.sql.SQLException
```

Devuelve el ID de colección para la conexión.

El método getDB2CurrentPackageSet solamente se aplica a conexiones con sistemas de bases de datos de DB2.

getDB2ProgressiveStreaming

Formato:

```
public int getDB2ProgressiveStreaming()
  throws java.sql.SQLException
```

Devuelve el valor de la modalidad continua actual para la conexión.

El valor devuelto depende de si la fuente de datos soporta la modalidad continua, de cómo se ha establecido la propiedad progressiveStreaming y de si se ha llamado a DB2Connection.setProgressiveStreaming:

- Si la fuente de datos no soporta la modalidad continua, siempre se devuelve 2 (NO), independientemente del valor de la propiedad progressiveStreaming.
- Si la fuente de datos soporta la modalidad continua y se ha llamado a DB2Connection.setProgressiveStreaming, el valor devuelto es el valor que ha establecido DB2Connection.setProgressiveStreaming.
- Si la fuente de datos soporta la modalidad continua y no se ha llamado a DB2Connection.setProgressiveStreaming, el valor devuelto es 2 (NO) si progressiveStreaming se ha establecido en DB2BaseDataSource.NO. Si progressiveStreaming se ha establecido en DB2BaseDataSource.YES o no se ha establecido, el valor devuelto es 1 (SÍ).

getDB2SecurityMechanism

Formato:

```
public int getDB2SecurityMechanism()
  throws java.sql.SQLException
```

Devuelve el mecanismo de seguridad que está en vigor para la conexión:

3 Seguridad por contraseña sin cifrar

4 Seguridad por ID de usuario solamente

7 Seguridad por contraseña cifrada

9 Seguridad por ID de usuario y contraseña cifrados

11 Kerberos, seguridad

12 Seguridad por ID de usuario y datos cifrados

13 Seguridad por ID de usuario, contraseña y datos cifrados

15 Seguridad por plugin

16 Seguridad por ID de usuario cifrado solamente

getDB2SystemMonitor

Formato:

```
public abstract DB2SystemMonitor getDB2SystemMonitor()
  throws java.sql.SQLException
```

Obtiene el objeto supervisor del sistema de la conexión. Cada conexión de IBM Data Server Driver para JDBC y SQLJ puede tener un supervisor del sistema individual.

getDBConcurrentAccessResolution

Formato:

```
public int getDBConcurrentAccessResolution()
  throws java.sql.SQLException
```

Devuelve la configuración de acceso simultáneo para la conexión. La configuración de acceso simultáneo se determina mediante el método setDBConcurrentAccessResolution o la propiedad concurrentAccessResolution.

getDBConcurrentAccessResolution se aplica únicamente a las conexiones con DB2 para z/OS y DB2 Database para Linux, UNIX y Windows.

getDBProgressiveStreaming

Formato:

```
public int getDB2ProgressiveStreaming()
  throws java.sql.SQLException
```

Devuelve el valor de la modalidad continua actual para la conexión.

El valor devuelto depende de si la fuente de datos soporta la modalidad continua, de cómo se ha establecido la propiedad progressiveStreaming y de si se ha llamado a DB2Connection.setProgressiveStreaming:

- Si la fuente de datos no soporta la modalidad continua, siempre se devuelve 2 (NO), independientemente del valor de la propiedad progressiveStreaming.
- Si la fuente de datos soporta la modalidad continua y se ha llamado a DB2Connection.setProgressiveStreaming, el valor devuelto es el valor que ha establecido DB2Connection.setProgressiveStreaming.
- Si la fuente de datos soporta la modalidad continua y no se ha llamado a DB2Connection.setProgressiveStreaming, el valor devuelto es 2 (NO) si progressiveStreaming se ha establecido en DB2BaseDataSource.NO. Si progressiveStreaming se ha establecido en DB2BaseDataSource.YES o no se ha establecido, el valor devuelto es 1 (SÍ).

getDBStatementConcentrator

Formato:

```
public int getDBStatementConcentrator()
  throws java.sql.SQLException
```

Devuelve la configuración de uso del concentrador de sentencias para la conexión. La configuración de uso del concentrador de sentencias se determina mediante el método setDBStatementConcentrator o la propiedad statementConcentrator.

getJccLogWriter

Formato:

```
public PrintWriter getJccLogWriter()
  throws java.sql.SQLException
```

Obtiene el destino de rastreo actual para la función de rastreo de IBM Data Server Driver para JDBC y SQLJ.

getJccSpecialRegisterProperties

Formato:

```
public java.util.Properties getJccSpecialRegisterProperties()
  throws java.sql.SQLException
```

Devuelve un objeto java.util.Properties, en el que las claves son los registros especiales que se admiten en la fuente de datos de destino, y los valores de clave son los valores actuales de esos registros especiales.

Este método no es aplicable a conexiones con fuentes de datos IBM Informix.

getSavePointUniqueOption

Formato:

```
public boolean getSavePointUniqueOption()
  throws java.sql.SQLException
```

Devuelve `true` si setSavePointUniqueOption se ha invocado más recientemente con un valor de `true`. En caso contrario devuelve `false`.

installDB2JavaStoredProcedure

Formato:

```
public void DB2Connection.installDB2JavaStoredProcedure(
  java.io.InputStream jarFile,
  int jarFileLength,
  String jarId)
  throws java.sql.SQLException
```

Invoca el procedimiento almacenado sqlj.install_jar en un servidor DB2 Database para Linux, UNIX y Windows para crear una nueva definición de un archivo JAR en el catálogo para dicho servidor.

Descripciones de parámetros:

jarFile
El contenido del archivo JAR debe definirse en el servidor.

jarFileLength
La longitud del archivo JAR debe definirse en el servidor.

jarId
Nombre para el JAR en la base de datos, en el formato *schema.JAR-idR* o *JAR-id*. Éste es el nombre que se utiliza para hacer referencia al archivo JAR de las sentencias de SQL. Si no especifica *esquema*, el sistema de bases de datos utiliza el ID de autorización de SQL contenido en el registro especial CURRENT SCHEMA. El propietario del archivo JAR es el ID de autorización del registro especial CURRENT SQLID.

Este método no es aplicable a conexiones con fuentes de datos IBM Informix.

isDB2Alive

Formato:

```
public boolean DB2Connection.isDB2Alive()
  throws java.sql.SQLException
```

Devuelve `true` si el socket de una conexión con la fuente de datos está todavía activo.

Importante: isDB2Alive está en desuso en la implementación para JDBC 4.0 de IBM Data Server Driver para JDBC y SQLJ. En su lugar, utilice Connection.isDBValid.

isDBValid

Formato:

```
public boolean DB2Connection.isDBValid(boolean throwException, int timeout)
  throws java.sql.SQLException
```

Devuelve `true` (verdadero) si la conexión no se ha cerrado y aún es válida. En caso contrario devuelve `false`.

Descripciones de parámetros:

throwException
Especifica si isDBValid lanza una SQLException si la conexión no es válida. Los valores posibles son:

true isDBValid lanza una SQLException si la conexión no es válida.

false isDBValid genera una SQLException sólo si el valor de *timeout* no es válido.

timeout
Tiempo en segundos que se debe esperar que a finalice la operación de base de datos que el controlador envía. El controlador somete la operación de base de datos a una fuente de datos para validar la conexión. Si el

periodo de tiempo de espera excedido caduca antes de que finalice la operación del la base de datos, isDBValid devuelve `false`. Un valor de 0 indica que no se ha producido ningún periodo de tiempo de espera excedido para la operación de base de datos.

Para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4, isDBValid genera una SQLException si el valor de *timeout* es inferior a 0.

Para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2, isDBValid genera una SQLException si el valor de *timeout* no es igual a 0.

Este método no es aplicable a conexiones con fuentes de datos IBM Informix.

prepareDB2OptimisticLockingQuery

Formato:

```
public java.sql.PreparedStatement
  DB2Connection.prepareDB2OptimisticLockingQuery(String sql,
  int returnOptimisticLockingColumns)
  throws SQLException
```

Crea un objeto PreparedStatement que puede solicitar información de bloqueo optimista.

Descripciones de parámetros:

sql
Sentencia de SQL que se debe preparar.

returnOptimisticLockingColumns
Especifica si se devuelven columnas de bloqueo optimista. Los valores posibles son:

Tabla 111. Valores para el parámetro returnOptimisticLockingColumns

Valor	Descripción
DB2Statement.RETURN_OPTLOCK_COLUMN_NONE (0)	No devolver columnas de bloqueo optimista.
DB2Statement.RETURN_OPTLOCK_COLUMN_ALWAYS (1)	Añade columnas de cambio de fila al conjunto de resultados incluso si no representan de forma exclusiva una única fila. Este valor es equivalente al atributo de preparación de base de datos WITH ROW CHANGE COLUMNS POSSIBLY DISTINCT.
DB2Statement.RETURN_OPTLOCK_COLUMN_NO_FALSE_NEGATIVES (2)	Añade columnas de cambio de fila al conjunto de resultados únicamente si representan una única fila. Este valor es equivalente al atributo de preparación de base de datos WITH ROW CHANGE COLUMNS ALWAYS DISTINCT.

reconfigureDB2Connection

Formato:

```
public void reconfigureDB2Connection(java.util.Properties properties)
  throws SQLException
```

Vuelve a configurar una conexión con valores nuevos. La conexión no debe devolverse a una agrupación de conexiones antes de que se vuelva a configurar. Se puede llamar a este método mientras una transacción se encuentra en curso y puede utilizarse para las conexiones fiables o no fiables.

Las conexiones fiables están soportadas para:

- IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 para:
  - DB2 Database para Linux, UNIX y Windows Versión 9.5 o versiones posteriores

- DB2 para z/OS Versión 9.1 o versiones posteriores
- IBM Informix Versión 11.70 o versiones posteriores

• IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS Versión 9.1 o versiones posteriores

Descripciones de parámetros:

properties
Propiedades nuevas de la conexión. Estas propiedades alteran temporalmente todas las propiedades que ya se han definido en la instancia de DB2Connection.

registerDB2XmlSchema

Formatos:

```
public void registerDB2XmlSchema(String[] sqlIdSchema,
  String[] sqlIdName,
  String[] xmlSchemaLocations,
  InputStream[] xmlSchemaDocuments,
  int[] xmlSchemaDocumentsLengths,
  InputStream[] xmlSchemaDocumentsProperties,
  int[] xmlSchemaDocumentsPropertiesLengths,
  InputStream xmlSchemaProperties,
  int xmlSchemaPropertiesLength,
  boolean isUsedForShredding)
  throws SQLException
public void registerDB2XmlSchema(String[] sqlIdSchema,
  String[] sqlIdName,
  String[] xmlSchemaLocations,
  String[] xmlSchemaDocuments,
  String[] xmlSchemaDocumentsProperties,
  String xmlSchemaProperties,
  boolean isUsedForShredding)
  throws SQLException
```

Registra un esquema XML con uno o más documentos de esquema XML. Si se procesan varios documentos de esquema XML con una sola llamada a registerDB2XmlSchema, dichos documentos se procesarán como parte de una sola transacción.

El primer formato de registerDB2XmlSchema es para documentos de esquema XML que se leen desde una corriente de entrada. El segundo formato de registerDB2XmlSchema es para documentos de esquema XML que se leen desde series.

Descripciones de parámetros:

sqlIdSchema
Nombre del esquema SQL para el esquema XML. Sólo se utiliza el primer elemento de la matriz *sqlIdSchema*. *sqlIdSchema* es un valor de serie con una longitud máxima de 128 bytes. El valor de *sqlIdSchema* debe seguir las reglas de denominación aplicables a cualquier nombre de esquema de SQL. El nombre no puede empezar por la serie 'SYS'. Si el valor de *sqlIdSchema* es nulo, el sistema de bases de datos utilizará el valor del registro especial CURRENT SCHEMA.

sqlIdName
Nombre SQL del esquema XML. Sólo se utiliza el primer elemento de la matriz *sqlIdName*. *sqlIdName* es un valor de serie con una longitud máxima de 128 bytes. El valor de *sqlIdName* debe seguir las reglas aplicables a un identificador de SQL. Si el valor de *sqlIdSchema* es nulo, el valor de *sqlIdName* puede ser nulo. En este caso, el sistema de bases de datos genera el valor de *sqlIdName*.

xmlSchemaLocations
Ubicaciones de esquema XML de los documentos de esquema XML principal de los esquemas que se están registrando. Los valores de ubicación del esquema XML suelen tener el formato URI. Cada valor *xmlSchemaLocations* es un valor de serie con una longitud máxima de 1.000 bytes. El valor se utiliza sólo para que coincida con la información que se especifica en el documento de esquema XML que hace referencia a dicho documento. El sistema de bases de datos no realiza ninguna validación de formato ni intenta resolver el URI.

xmlSchemaDocuments
Contenido de los documentos de esquema XML principal. Cada valor *xmlSchemaDocuments* es un valor de serie o de corriente de entrada con una longitud máxima de 30. Los valores no pueden ser nulos.

xmlSchemaDocumentsLengths
Longitudes de los documentos de esquema XML del parámetro *xmlSchemaDocuments*, en el caso de que se utilice el primer formato de registerDB2XmlSchema. Cada valor *xmlSchemaDocumentsLengths* es un valor int.

xmlSchemaDocumentsProperties
Contiene propiedades de los documentos de esquema XML principal, como las propiedades que utiliza un sistema de creación de versiones del esquema XML externo. El sistema de bases de datos no realiza ninguna validación del contenido de dichos valores. Se almacenan en la tabla XSR para su recuperación y se utilización en otras herramientas e implementaciones de depósito de esquemas XML. Cada valor *xmlSchemaDocumentsProperties* es un valor de serie o de corriente de entrada con una longitud máxima de 5 MB. Si no debe pasarse ninguna propiedad, el valor será nulo.

xmlSchemaDocumentsPropertiesLengths
Longitudes de las propiedades de esquema XML del parámetro *xmlSchemaDocumentsProperties*, en el caso de que se utilice el primer formato de registerDB2XmlSchema. Cada valor *xmlSchemaDocumentsPropertiesLengths* es un valor int.

xmlSchemaProperties
Contiene propiedades de todo el documento XML, como las que utiliza un sistema de creación de versiones del esquema XML externo. El sistema de bases de datos no realiza ninguna validación del contenido de este valor. Se almacenan en la tabla XSR para su recuperación y se utilización en otras herramientas e implementaciones de depósito de esquemas XML. El valor *xmlSchemaProperties* es un valor de serie o de corriente de entrada con una longitud máxima de 5 MB. Si no debe pasarse ninguna propiedad, el valor será nulo.

xmlSchemaPropertiesLengths
Longitud de la propiedad del esquema XML del parámetro *xmlSchemaProperties*, en el caso de que se utilice el primer formato de registerDB2XmlSchema. El valor *xmlSchemaPropertiesLengths* es un valor int.

isUsedForShredding
Indica si existen anotaciones en el esquema que debe utilizarse para la descomposición XML. *isUsedForShredding* es un valor booleano.

Este método no es aplicable a conexiones con fuentes de datos IBM Informix.

setDBConcurrentAccessResolution

Formato:

```
public void setDBConcurrentAccessResolution(int concurrentAccessResolution)
  throws java.sql.SQLException
```

Especifica si IBM Data Server Driver para JDBC y SQLJ solicita que una transacción de lectura pueda acceder a una imagen confirmada y consistente de filas que están bloqueadas por transacciones de grabación de forma incompatible, si la fuente de datos soporta el acceso a los datos confirmados actualmente y si el nivel de aislamiento de la aplicación es estabilidad de cursor (CS) o estabilidad de lectura (RS). Esta opción tiene el mismo efecto que la opción de vinculación CONCURRENTACCESSRESOLUTION de DB2. setDBConcurrentAccessResolution afecta únicamente a las sentencias que se crean tras ejecutarse setDBConcurrentAccessResolution.

setDBConcurrentAccessResolution se aplica únicamente a las conexiones con DB2 para z/OS y DB2 Database para Linux, UNIX y Windows.

Descripciones de parámetros:

concurrentAccessResolution
Uno de los valores enteros siguientes:

DB2BaseDataSource.-CONCURRENTACCESS_USE_CURRENTLY_COMMITTED (1)
IBM Data Server Driver para JDBC y SQLJ solicita lo siguiente:

- Las transacciones de lectura deben acceder a los datos confirmados actualmente cuando se están actualizando o suprimiendo los datos.
- Las transacciones de lectura deben omitir las filas que se están insertando.

DB2BaseDataSource.CONCURRENTACCESS_WAIT_FOR_OUTCOME (2)
IBM Data Server Driver para JDBC y SQLJ solicita lo siguiente:

- Las transacciones de lectura deben esperar a una operación de confirmación o retrotracción cuando encuentran datos que se están actualizando o suprimiendo.
- Las transacciones de lectura no deben omitir las filas que se están insertando.

DB2BaseDataSource.CONCURRENTACCESS_NOT_SET (0)
Habilita el comportamiento por omisión del servidor de datos para las transacciones de lectura cuando se produce una contención de bloqueo. Éste es el valor por omisión.

setDBProgressiveStreaming

Formato:

```
public void setDB2ProgressiveStreaming(int newSetting)
  throws java.sql.SQLException
```

Establece el valor de la modalidad continua para todos los objetos de ResultSet que se crean en la conexión.

Descripciones de parámetros:

newSetting
El valor de la nueva modalidad continua. Los valores posibles son:

DB2BaseDataSource.YES (1)
Habilitar la modalidad continua. Si la fuente de datos no soporta la modalidad continua, el valor no entrará en vigor.

DB2BaseDataSource.NO (2)
Inhabilitar la modalidad continua.

setDBStatementConcentrator

Formato:

```
public void setDBStatementConcentrator(int statementConcentratorUse)
  throws java.sql.SQLException
```

Especifica si IBM Data Server Driver para JDBC y SQLJ emplea la funcionalidad del concentrador de sentencias de la fuente de datos. El concentrador de sentencias es la capacidad de eludir la preparación de una sentencia cuando es igual que una sentencia en la antememoria de sentencias dinámicas, excepto los valores literales. La funcionalidad del concentrador de sentencias se aplica únicamente a las sentencias de SQL que tienen literales pero no tienen marcadores de parámetro. setDBStatementConcentrator prevalece sobre el valor de la propiedad Connection o DataSource de statementConcentrator. setDBStatementConcentrator afecta únicamente a las sentencias que se crean tras ejecutarse setDBStatementConcentrator.

Descripciones de parámetros:

statementConcentratorUse
Uno de los valores enteros siguientes:

DB2BaseDataSource.STATEMENT_CONCENTRATOR_OFF (1)
IBM Data Server Driver para JDBC y SQLJ no utiliza la funcionalidad del concentrador de sentencias de la fuente de datos.

DB2BaseDataSource.STATEMENT_CONCENTRATOR_WITH_LITERALS (2)
IBM Data Server Driver para JDBC y SQLJ emplea la funcionalidad de concentrador de sentencias de la fuente de datos.

DB2BaseDataSource.STATEMENT_CONCENTRATOR_NOT_SET (0)
Habilita el comportamiento por omisión del servidor de datos para la funcionalidad del concentrador de sentencias. Éste es el valor por omisión.

En el caso de las fuentes de datos DB2 Database para Linux, UNIX y Windows que dan soporte a la funcionalidad del concentrador de sentencias, la funcionalidad se emplea si el parámetro de configuración STMT_CONC está establecido en ON en la fuente de datos. En caso contrario, no se utiliza la funcionalidad del concentrador de sentencias.

En el caso de las fuentes de datos DB2 para z/OS que dan soporte a la funcionalidad del concentrador de sentencias, la funcionalidad no se utiliza si no se ha determinado statementConcentrator.

removeDB2JavaStoredProcedure

Formato:

```
public void DB2Connection.removeDB2JavaStoredProcedure(
  String jarId)
  throws java.sql.SQLException
```

Invoca el el procedimiento almacenado sqlj.remove_jar en un servidor DB2 Database para Linux, UNIX y Windows para suprimir la definición de un archivo JAR en el catálogo de ese servidor.

Descripciones de parámetros:

jarId
Nombre para el JAR en la base de datos, en el formato *schema.JAR-idR* o *JAR-id*. Éste es el nombre que se utiliza para hacer referencia al archivo JAR de las sentencias de SQL. Si no especifica *esquema*, el sistema de bases de datos utiliza el ID de autorización de SQL contenido en el registro especial CURRENT SCHEMA.

Este método no es aplicable a conexiones con fuentes de datos IBM Informix.

replaceDB2JavaStoredProcedure

Formato:

```
public void DB2Connection.replaceDB2JavaStoredProcedure(
  java.io.InputStream jarFile,
  int jarFileLength,
  String jarId)
  throws java.sql.SQLException
```

Invoca el procedimiento almacenado sqlj.replace_jar en un servidor DB2 Database para Linux, UNIX y Windows para sustituir la definición de un archivo JAR en el catálogo de ese servidor.

Descripciones de parámetros:

jarFile
Contenido del archivo JAR que debe sustituirse en el servidor.

jarFileLength
Longitud del archivo JAR que debe sustituirse en el servidor.

jarId
Nombre para el JAR en la base de datos, en el formato *schema.JAR-idR* o *JAR-id*. Éste es el nombre que se utiliza para hacer referencia al archivo JAR de las sentencias de SQL. Si no especifica *esquema*, el sistema de bases de datos utiliza el ID de autorización de SQL contenido en el registro especial CURRENT SCHEMA. El propietario del archivo JAR es el ID de autorización del registro especial CURRENT SQLID.

Este método no es aplicable a conexiones con fuentes de datos IBM Informix.

reuseDB2Connection (trusted connection reuse)

Formatos:

```
public void reuseDB2Connection(byte[] cookie,
  String user,
  String password,
  String usernameRegistry,
  byte[] userSecToken,
  String originalUser,
  java.util.Properties properties)
  throws java.sql.SQLException
public void reuseDB2Connection(byte[] cookie,
  org.ietf.GSSCredential gssCredential,
  String usernameRegistry,
  byte[] userSecToken,
  String originalUser,
  java.util.Properties properties)
  throws java.sql.SQLException
```

Las conexiones fiables están soportadas para:

- IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 para:
  - DB2 Database para Linux, UNIX y Windows Versión 9.5 o versiones posteriores

  - DB2 para z/OS Versión 9.1 o versiones posteriores
  - IBM Informix Versión 11.70 o versiones posteriores
- IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS Versión 9.1 o versiones posteriores

El segundo de estos formatos de reuseDB2Connection no se aplica a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS.

Un servidor de aplicaciones fiable utiliza estos formatos de reuseDB2Connection para volver a utilizar una conexión fiable preexistente en nombre de un usuario nuevo. Se pasan las propiedades que pueden restaurarse, incluido el ID de usuario nuevo. El servidor de bases de datos restaura la conexión física asociada. Si reuseDB2Connection se ejecuta correctamente, el nuevo usuario podrá utilizar inmediatamente la conexión con propiedades diferentes.

Descripciones de parámetros:

cookie
: Cookie exclusivo creado por el controlador JDBC para la instancia de Connection. Sólo el servidor de aplicaciones y el controlador JDBC subyacente que estableció la conexión fiable inicial conocen la cookie. El servidor de aplicaciones pasa la cookie creada por el controlador en el momento de la creación de la instancia de la conexión agrupada. El controlador JDBC comprueba que la cookie proporcionada coincida con la cookie de la conexión física fiable subyacente para garantizar que la petición se ha creado a partir del servidor de aplicaciones que ha establecido la conexión física fiable. Si las cookies coinciden, el usuario podrá utilizar inmediatamente la conexión con propiedades diferentes.

user
: ID de cliente que el sistema de bases de datos utiliza para establecer el ID de autorización de la base de datos. Si el servidor de aplicaciones no ha autentificado al usuario, el servidor de aplicaciones deberá pasar un ID de cliente que represente a un usuario sin autentificación.

contraseña
: Contraseña de *user*.

gssCredential
: Si la fuente de datos utiliza la seguridad Kerberos, especifica una credencial delegada que se pasa desde otro principal.

userNameRegistry
: Nombre que identifica un servicio de correlación que correlaciona un ID de usuario de estación de trabajo con un ID de z/OS RACF. Un ejemplo de servicio de correlación es la correlación de identidades empresariales (EIM) de servicios de seguridad integrados. El servicio de correlación se define mediante un plugin. Los proveedores de plugins definen los valores válidos de *userNameRegistry*. Si el valor de *userNameRegistry* es nulo, no se realizará ninguna correlación de usuario *user*.

userSecToken
: Símbolos de seguridad del cliente. Este valor se rastrea como parte de los datos contables de DB2 para z/OS. El contenido de *userSecToken* es descrito por el servidor de aplicaciones y es especificado por el sistema de bases de datos como símbolo de seguridad del servidor de aplicaciones.

originalUser
: ID del usuario original utilizado por el servidor de aplicaciones.

properties
Propiedades de la conexión reutilizada.

reuseDB2Connection (reutilización no fiable con reautentificación)

Formatos:

```
public void reuseDB2Connection(String user,
  String password,
  java.util.Properties properties)
  throws java.sql.SQLException
public void reuseDB2Connection(
  org.ietf.jgss.GSSCredential gssCredential,
  java.util.Properties properties)
  throws java.sql.SQLException
```

El primero de estos formatos de reuseDB2Connection no es compatible para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS.

El segundo de estos formatos de reuseDB2Connection no se aplica a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS.

En un entorno de agrupación heterogéneo, estos formatos de reuseDB2Connection reutilizan una instancia de Connection existente después de la reautentificación.

Descripción de parámetros:

user
ID de autorización que se utiliza para establecer la conexión.

contraseña
Contraseña del ID de autorización que se utiliza para establecer la conexión.

gssCredential
Si la fuente de datos utiliza la seguridad Kerberos, especifica una credencial delegada que se pasa desde otro principal.

properties
Propiedades de la conexión reutilizada. Estas propiedades alteran temporalmente todas las propiedades que ya se han definido en la instancia de DB2Connection.

reuseDB2Connection (reutilización no fiable o fiable sin reautentificación)

Formatos:

```
public void reuseDB2Connection(java.util.Properties properties)
  throws java.sql.SQLException
```

Reutiliza una instancia de conexión sin reautentificación. Este método está diseñado para reutilizar la instancia de Connection en el caso de que las propiedades no cambien.

Las conexiones fiables están soportadas para:

- IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 para:
  - DB2 Database para Linux, UNIX y Windows Versión 9.5 o versiones posteriores
  - DB2 para z/OS Versión 9.1 o versiones posteriores
  - IBM Informix Versión 11.70 o versiones posteriores
- IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS Versión 9.1 o versiones posteriores

Este método sirve para la *dirty reuse* de una conexión. Esto significa que el estado de la conexión no se restablece cuando se reutiliza el objeto desde la agrupación. Los valores de propiedad y los valores de registro especiales siguen vigentes a menos que los alteren temporalmente las propiedades pasadas. No se suprimen las tablas temporales globales. Las propiedades que no se especifiquen no se volverán a inicializar. Todas las propiedades transitorias estándar de JDBC, como el nivel de aislamiento, la modalidad de confirmación automático y la modalidad de sólo lectura se restablecen a sus valores por omisión de JDBC. Ciertas propiedades, como por ejemplo, `user`, `password`, `databaseName`, `serverName`, `portNumber`, `planName` y `pkList` permanecerán sin modificar.

Descripción de parámetros:

properties
: Propiedades de la conexión reutilizada. Estas propiedades alteran temporalmente todas las propiedades que ya se han definido en la instancia de DB2Connection.

setDB2ClientAccountingInformation

Formato:

```
public void setDB2ClientAccountingInformation(String info)
  throws java.sql.SQLException
```

Especifica información contable para la conexión. Esta información se utiliza con fines de contabilidad de clientes. Este valor puede cambiar durante una conexión.

Descripción de parámetros:

info
: Información contable especificada por el usuario. La longitud máxima depende del servidor. Para un servidor DB2 Database para Linux, UNIX y Windows, la longitud máxima es de 255 bytes. Para un servidor DB2 para z/OS, la longitud máxima es de 22 bytes. Una serie vacía de Java ("") o un valor `null` de Java es válido para este parámetro.

Importante: setDB2ClientAccountingInformation está en desuso en la implementación para JDBC 4.0 de IBM Data Server Driver para JDBC y SQLJ. En su lugar utilice java.sql.Connection.setClientInfo.

setDB2ClientApplicationInformation

Formato:

```
public String setDB2ClientApplicationInformation(String info)
  throws java.sql.SQLException
```

Especifica la información sobre la aplicación del cliente actual.

Importante: setDB2ClientApplicationInformation está en desuso en la implementación para JDBC 4.0 de IBM Data Server Driver para JDBC y SQLJ. En su lugar utilice java.sql.Connection.setClientInfo.

Descripción de parámetros:

info
: Información sobre la aplicación especificada por el usuario. La longitud máxima depende del servidor. Para un servidor DB2 Database para Linux, UNIX y Windows, la longitud máxima es de 255 bytes. Para un servidor DB2 para z/OS, la longitud máxima es 32 bytes. Una serie vacía de Java ("") o un valor `null` de Java es válido para este parámetro.

setDB2ClientDebugInfo

Formatos:

```
public void setDB2ClientDebugInfo(String debugInfo)
  throws java.sql.SQLException
public void setDB2ClientDebugInfo(String mgrInfo,
  String traceInfo)
  throws java.sql.SQLException
```

Establece un valor para el atributo de conexión CLIENT DEBUGINFO para notificar al sistema de bases de datos que los procedimientos almacenados y las funciones definidas por el usuario que están utilizando la conexión se están ejecutando en modalidad de depuración. El atributo CLIENT DEBUGINFO es utilizado por el depurador unificado de DB2. Utilice el primer formato para establecer toda la serie de CLIENT DEBUGINFO. Utilice el segundo formato para modificar únicamente el gestor de sesiones y la información de rastreo de la serie CLIENT DEBUGINFO.

Para establecer el atributo CLIENT DEBUGINFO para una serie cuya longitud sea mayor que cero, es necesario contar con uno de los privilegios siguientes:
- El privilegio DEBUGSESSION
- Autorización SYSADM

Descripción de parámetros:

debugInfo
Cadena de un máximo de 254 bytes con el formato siguiente:

```
Mip:puerto,Iip,Pidp,Tidh,Cid,Lniv
```

Las partes de la serie son:

M*ip:puerto*
Dirección IP y número de puerto del gestor de la sesión

I*ip* Dirección IP de cliente

P*idp* ID de proceso de cliente

T*idh* ID de hebra de cliente (opcional)

C*id* ID generado por la conexión de datos

L*niv* Nivel de rastreo de diagnóstico de biblioteca de depuración

Por ejemplo:

```
M9.72.133.89:8355,I9.72.133.89,P4552,T123,C1,L0
```

Consulte la descripción de SET CLIENT DEBUGINFO para obtener detalles de esta serie.

mgrInfo
Serie con el formato siguiente, la cual especifica la dirección IP y el número de puerto del gestor de sesiones del depurador unificado.

```
Mip:puerto
```

Por ejemplo:

```
M9.72.133.89:8355
```

Consulte la descripción de SET CLIENT DEBUGINFO para obtener detalles de esta serie.

trcInfo
Serie con el formato siguiente, la cual especifica el nivel de rastreo de diagnóstico de la biblioteca de depuración.

L*niv*

Por ejemplo:

```
L0
```

Consulte la descripción de SET CLIENT DEBUGINFO para obtener detalles de esta serie.

setDB2ClientProgramId

Formato:

```
public abstract void setDB2ClientProgramId(String program-ID)
  throws java.sql.SQLException
```

Establece un identificador de programa definido por el usuario correspondiente a la conexión, en servidores DB2 para z/OS. Dicho identificador de programa es una serie de 80 bytes que se utiliza para identificar al llamador.

setDB2ClientProgramId no se aplica a los servidores de datos DB2 Database para Linux, UNIX y Windows ni IBM Informix.

El servidor DB2 para z/OS coloca la serie en los registros de rastreo IFCID 316 junto con los demás datos estadísticos, de modo que se pueda identificar el programa que está asociado con una sentencia de SQL determinada.

setDB2ClientUser

Formato:

```
public void setDB2ClientUser(String user)
  throws java.sql.SQLException
```

Especifica el nombre de usuario del cliente actual de la conexión. Este nombre se utiliza con fines de contabilidad de clientes, y no es el valor de usuario (user) de la conexión JDBC. A diferencia del valor de usuario de la conexión JDBC, el nombre de usuario del cliente actual puede cambiar durante una conexión.

Descripción de parámetros:

user
ID de usuario del cliente actual. La longitud máxima depende del servidor. Para un servidor DB2 Database para Linux, UNIX y Windows, la longitud máxima es de 255 bytes. Para un servidor DB2 para z/OS, la longitud máxima es de 16 bytes. Una serie vacía de Java ("") o un valor `null` de Java es válido para este parámetro.

Importante: setDB2ClientUser está en desuso en la implementación para JDBC 4.0 de IBM Data Server Driver para JDBC y SQLJ. En su lugar utilice java.sql.Connection.setClientInfo.

setDB2ClientWorkstation

Formato:

```
public void setDB2ClientWorkstation(String name)
  throws java.sql.SQLException
```

Especifica el nombre de estación de trabajo del cliente actual de la conexión. Este nombre se utiliza con fines de contabilidad de clientes. El nombre de estación de trabajo del cliente actual puede cambiar durante una conexión.

Descripción de parámetros:

name
Nombre de la estación de trabajo para el cliente actual. La longitud máxima depende del servidor. Para un servidor DB2 Database para Linux, UNIX y Windows, la longitud máxima es de 255 bytes. Para un servidor DB2 para z/OS, la longitud máxima es de 18 bytes. Una serie vacía de Java ("") o un valor `null` de Java es válido para este parámetro.

Importante: getDB2ClientWorkstation está en desuso en la implementación para JDBC 4.0 de IBM Data Server Driver para JDBC y SQLJ. En su lugar utilice java.sql.Connection.getClientInfo.

setDB2CurrentPackagePath

Formato:

```
public void setDB2CurrentPackagePath(String packagePath)
  throws java.sql.SQLException
```

Especifica una lista de ID de colección en que el sistema de bases de datos busca paquetes JDBC y SQLJ.

El método setDB2CurrentPackagePath solamente se aplica a conexiones con sistemas de bases de datos de DB2.

Descripción de parámetros:

packagePath
Lista de ID de colección, separados por comas.

setDB2CurrentPackageSet

Formato:

```
public void setDB2CurrentPackageSet(String packageSet)
  throws java.sql.SQLException
```

Especifica el ID de colección de la conexión. Cuando define este valor, también define el ID de colección de la instancia de IBM Data Server Driver para JDBC y SQLJ que se utiliza para la conexión.

El método setDB2CurrentPackageSet solamente se aplica a conexiones con sistemas de bases de datos de DB2.

Descripción de parámetros:

packageSet
ID de colección para la conexión. La longitud máxima del valor de *packageSet* es 18 bytes. Puede invocar este método como alternativa a ejecutar la sentencia SET CURRENT PACKAGESET de SQL en su programa.

setDB2ProgressiveStreaming

Formato:

```
public void setDB2ProgressiveStreaming(int newSetting)
  throws java.sql.SQLException
```

Establece el valor de la modalidad continua para todos los objetos de ResultSet que se crean en la conexión.

Descripciones de parámetros:

newSetting
El valor de la nueva modalidad continua. Los valores posibles son:

DB2BaseDataSource.YES (1)
Habilitar la modalidad continua. Si la fuente de datos no soporta la modalidad continua, el valor no entrará en vigor.

DB2BaseDataSource.NO (2)
Inhabilitar la modalidad continua.

setJccLogWriter

Formatos:

```
public void setJccLogWriter(PrintWriter logWriter)
  throws java.sql.SQLException

public void setJccLogWriter(PrintWriter logWriter, int traceLevel)
  throws java.sql.SQLException
```

Habilita o inhabilita la función de rastreo de IBM Data Server Driver para JDBC y SQLJ, o cambia el destino de rastreo durante una conexión activa.

Descripciones de parámetros:

logWriter
Objeto de tipo java.io.PrintWriter en que IBM Data Server Driver para JDBC y SQLJ graba la salida del rastreo. Para desactivar el rastreo, establezca el valor de *logWriter* en `null`.

traceLevel
Especifica los tipos de rastreos que se deben recoger. Consulte la descripción de la propiedad *traceLevel* en el tema sobre propiedades de IBM Data Server Driver para JDBC y SQLJ para conocer los valores válidos.

setSavePointUniqueOption

Formato:

```
public void setSavePointUniqueOption(boolean distintivo)
  throws java.sql.SQLException
```

Especifica si una aplicación puede reutilizar un nombre de punto de salvaguarda dentro de una unidad de recuperación. Los valores posibles son:

true Un método Connection.setSavepoint(savepoint-name) no puede especificar el mismo valor para *nombre-puntosalvaguarda* más de una vez dentro de la misma unidad de recuperación.

false Un método Connection.setSavepoint(savepoint-name) puede especificar el mismo valor para *nombre-puntosalvaguarda* más de una vez dentro de la misma unidad de recuperación.

Cuando se ha especificado `false`, si se ejecuta el método Connection.setSavepoint(savepoint-name) y ya existe un punto de salvaguarda con el nombre *nombre-puntosalvaguarda* dentro de la unidad de recuperación, el gestor de la base de datos destruye el punto de salvaguarda existente y crea uno nuevo con el mismo *nombre-puntosalvaguarda*.

Reutilizar un punto de salvaguarda no equivale a ejecutar Connection.releaseSavepoint(savepoint-name). Connection.releaseSavepoint(savepoint-name) libera el *nombre-puntosalvaguarda* y todos los puntos de salvaguarda que se fijaron posteriormente.

updateDB2XmlSchema

Formato:

```
public void updateDB2XmlSchema(String[] targetSqlIdSchema,
  String[] targetSqlIdName,
  String[] sourceSqlIdSchema,
  String[] sourceSqlIdName,
  String[] xmlSchemaLocations,
  boolean dropSourceSchema)
  throws SQLException
```

Actualiza el contenido de un esquema XML con el contenido de otro esquema XML del depósito de esquemas XML, y opcionalmente elimina el esquema de origen. Los documentos de esquema contenidos en el esquema XML de destino son sustituidos por los documentos de esquema del esquema XML de origen. Antes de invocar updateDB2XmlSchema, es necesario registrar los esquemas XML de origen y destino.

Es necesario el privilegio ALTERIN de SQL para actualizar el esquema XML de destino. Es necesario el privilegio DROPIN de SQL para eliminar el esquema XML de origen.

Descripciones de parámetros:

targetSqlIdSchema
: Nombre del esquema SQL correspondiente a un esquema XML registrado que se debe actualizar. *targetSqlIdSchema* es un valor de serie con una longitud máxima de 128 bytes.

targetSqlIdName
: Nombre del esquema XML registrado que se debe actualizar. *targetSqlIdName* es un valor de serie con una longitud máxima de 128 bytes.

sourceSqlIdSchema
: Nombre del esquema de SQL correspondiente a un esquema XML registrado que se utiliza para actualizar el esquema XML de destino. *sourceSqlIdSchema* es un valor de serie con una longitud máxima de 128 bytes.

sourceSqlIdName
: Nombre del esquema XML registrado que se utiliza para actualizar el esquema XML de destino. *sourceSqlIdName* es un valor de serie con una longitud máxima de 128 bytes.

dropSourceSchema
: Indica si el esquema XML de origen se debe eliminar después de actualizar el esquema XML de destino. *dropSourceSchema* es un valor booleano. `false` es el valor por omisión.

Este método no es aplicable a conexiones con fuentes de datos IBM Informix.

Clase DB2ConnectionPoolDataSource

DB2ConnectionPoolDataSource es una fábrica para los objetos PooledConnection. Un objeto que implementa esta interfaz se registra con un servicio de asignación de nombres que se basa en Java Naming and Directory Interface (JNDI).

La clase com.ibm.db2.jcc.DB2ConnectionPoolDataSource amplía la clase com.ibm.db2.jcc.DB2BaseDataSource e implementa las interfaces javax.sql.ConnectionPoolDataSource, java.io.Serializable y javax.naming.Referenceable.

Propiedades de DB2ConnectionPoolDataSource

Estas propiedades sólo se definen para el IBM Data Server Driver para JDBC y SQLJ. Consulte "Propiedades de IBM Data Server Driver para JDBC y SQLJ" para obtener una explicación de estas propiedades.

Estas propiedades tienen un método setXXX para establecer el valor de la propiedad y un método getXXX para recuperar el valor. Un método setXXX tiene este formato:

```
void setNombre-propiedad(tipo-datos valor-propiedad)
```

Un método getXXX tiene este formato:

```
tipo-datos getNombre-propiedad()
```

Nombre-propiedad es el nombre de propiedad no calificado, con el primer carácter escrito en mayúsculas.

La tabla siguiente lista las propiedades de IBM Data Server Driver para JDBC y SQLJ y los tipos de datos asociados a ellas.

Tabla 112. Propiedades de DB2ConnectionPoolDataSource y tipos de datos asociados a ellas

Nombre de propiedad	Tipo de datos
com.ibm.db2.jcc.DB2ConnectionPoolDataSource.maxStatements	int

Métodos DB2ConnectionPoolDataSource

getDB2PooledConnection

Formatos:

```
public DB2PooledConnection getDB2PooledConnection(String usuario,
  String password,
  java.util.Properties properties)
  throws java.sql.SQLException
public DB2PooledConnection getDB2PooledConnection(
  org.ietf.jgss.GSSCredential gssCredential,
  java.util.Properties properties)
  throws java.sql.SQLException
```

Establece la conexión no fiable inicial en un entorno de agrupación heterogéneo.

El primer formato de getDB2PooledConnection ofrece un ID de usuario y una contraseña. El segundo formato de getDB2PooledConnection es para conexiones que utilizan la seguridad Kerberos.

Descripciones de parámetros:

user
ID de autorización que se utiliza para establecer la conexión.

contraseña
Contraseña del ID de autorización que se utiliza para establecer la conexión.

gssCredential
Si la fuente de datos utiliza la seguridad Kerberos, especifica una credencial delegada que se pasa desde otro principal.

propiedades
Propiedades de la conexión.

getDB2TrustedPooledConnection

Formatos:

```
public Object[] getDB2TrustedPooledConnection(String usuario,
  String password,
  java.util.Properties properties)
  throws java.sql.SQLException
public Object[] getDB2TrustedPooledConnection(
  java.util.Properties properties)
  throws java.sql.SQLException
public Object[] getDB2TrustedPooledConnection(
  org.ietf.jgss.GSSCredential gssCredential,
  java.util.Properties properties)
  throws java.sql.SQLException
```

Un servidor de aplicaciones que utiliza un ID de autorización del sistema utiliza este método para establecer una conexión fiable.

Las conexiones fiables están soportadas para:

- IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 para:
  - DB2 Database para Linux, UNIX y Windows Versión 9.5 o versiones posteriores
  - DB2 para z/OS Versión 9.1 o versiones posteriores
  - IBM Informix Versión 11.70 o versiones posteriores
- IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS Versión 9.1 o versiones posteriores

Los elementos siguientes se devuelven en Object[]:

- El primer elemento es una instancia de DB2PooledConnection fiable.
- El segundo elemento es una cookie exclusiva para la instancia de conexión agrupada que se ha generado.

El primer formato de getDB2TrustedPooledConnection ofrece un ID de usuario y una contraseña, mientras que el segundo formato de getDB2TrustedPooledConnection utiliza el ID de usuario y la contraseña del objeto DB2ConnectionPoolDataSource. El tercer formato de getDB2TrustedPooledConnection es para conexiones que utilizan la seguridad Kerberos.

Descripciones de parámetros:

user
ID de autorización de DB2 que se utiliza para establecer la conexión fiable con el servidor de bases de datos.

contraseña
Contraseña del ID de autorización que se utiliza para establecer la conexión fiable.

gssCredential
Si la fuente de datos utiliza la seguridad Kerberos, especifica una credencial delegada que se pasa desde otro principal.

propiedades
Propiedades de la conexión.

Interfaz DB2DatabaseMetaData

La interfaz com.ibm.db2.jcc.DB2DatabaseMetaData amplía la interfaz java.sql.DatabaseMetaData.

Métodos DB2DatabaseMetaData

Los métodos siguientes se definen únicamente para IBM Data Server Driver para JDBC y SQLJ.

getDBFunctionColumns

Formato:

```
public java.sql.ResultSet getDBFunctionColumns(
  String catálogo,
  String patrón_esquema,
  String patrón_módulo,
  String patrón_nombre_función,
  String patrón_nombre_columna)
  throws java.sql.SQLException
```

Devuelve un objeto ResultSet que describe los parámetros y valores de retorno para las funciones definidas por el usuario o las funciones incorporadas definidas en una fuente de datos. Los valores de retorno y los parámetros de función para los que se devuelve información coinciden con el nombre de catálogo, el patrón de esquema, el patrón de módulo, el patrón de nombre de función y el patrón de nombre de columna especificados por los parámetros de entrada. ResultSet contiene las columnas siguientes:

Número de columna	Nombre de columna	Tipo de datos	Descripción
1	FUNCTION_CAT	Serie	Nombre del catálogo. El valor es nulo si esta función no tiene catálogos.
2	FUNCTION_SCHEM	Serie	Nombre del esquema que contiene FUNCTION_NAME. Este valor puede ser nulo.
3	FUNCTION_NAME	Serie	Nombre de la función.
4	COLUMN_NAME	Serie	Nombre del parámetro de función.
5	COLUMN_TYPE	Short	Tipo de parámetro: • functionColumnUnknown (0): desconocido • functionColumnIn (1): entrada • functionColumnInOut (2): entrada o salida • functionColumnResult (3): columna de conjunto de resultados • functionColumnOut (4): salida • functionColumnReturn (5): valor devuelto
6	DATA_TYPE	int	Tipo de datos de SQL del parámetro identificado mediante COLUMN_NAME.
7	TYPE_NAME	Serie	Serie de caracteres que representa el nombre del tipo de datos correspondiente a DATA_TYPE.
8	PRECISION	int	Precisión de un parámetro numérico exacto.
9	LENGTH	int	Longitud del parámetro.
10	SCALE	short	Escala de un parámetro numérico exacto.
11	RADIX	short	10, 2 o nulo: • Si DATA_TYPE es un tipo de datos numérico aproximado, este parámetro contiene el valor 2. • Si DATA_TYPE es un tipo de datos numérico exacto, este parámetro contiene el valor 10. • Para los tipos de datos en los que no es aplicable la base, este valor es nulo.

Número de columna	Nombre de columna	Tipo de datos	Descripción
12	NULLABLE	short	Especifica si el parámetro puede contener valores NULL: **functionNoNulls (0)** El parámetro no puede contener valores NULL. **functionNullable (1)** El parámetro puede contener valores NULL. **functionNullableUnknown (2)** No se sabe si el parámetro puede ser nulo.
13	REMARKS	Serie	Información descriptiva sobre el parámetro o nulo.
14	CHAR_OCTET_LENGTH	int	Para los tipos de datos de carácter, el número máximo de bytes en el parámetro. Para el tipo XML, se devuelve cero. Para el resto de los tipos de datos, este valor es nulo.
15	ORDINAL_POSITION	int	Posición del parámetro, empezando en 1. Este valor es 0 si COLUMN_TYPE es 5 (valor de retorno).
16	IS_NULLABLE	Serie	Contiene la serie 'NO' si se sabe que el parámetro no tiene posibilidad de ser nulo, 'YES' si el parámetro puede ser nulo o una serie vacía si no se sabe.
17	SPECIFIC_NAME	Serie	Nombre que identifica de forma exclusiva a la función dentro de su esquema.
18	FUNCTION_MODULE	Serie	Nombre del módulo que contiene la función. Este valor puede ser nulo.

Descripciones de parámetros:

catálogo
Una serie vacía o '%'.

schemaName
Especifica el nombre de esquema (calificador) de los objetos para los que se va a devolver información.

El valor puede identificar un nombre de esquema único o más de uno, incluyendo el carácter de porcentaje de coincidencia de patrones (%).

patrón_módulo
Especifica el nombre del módulo para el que se va a devolver información de parámetro.

El valor puede identificar un nombre de módulo único o más de uno, incluyendo el carácter de porcentaje de coincidencia de patrones (%).

patrón_nombre_función
Especifica el nombre de la función para la que se va a devolver información de parámetro.

El valor puede identificar un nombre de esquema único, o puede identificar más de un nombre de procedimiento incluyendo el carácter de porcentaje de coincidencia de patrones (%).

patrón_nombre_columna
Especifica los parámetros para los que se va a devolver información de parámetro.

El valor puede identificar un nombre de esquema único, o puede identificar más de un nombre de parámetro incluyendo el carácter de porcentaje de coincidencia de patrones (%).

Este método solamente se aplica a conexiones con fuentes de datos de DB2 Database para Linux, UNIX y Windows 9.7 o posteriores. Se emite una SqlFeatureNotSupportedException si la fuente de datos es otra fuente de datos distinta.

getDBFunctions

Formato:

```
public java.sql.ResultSet getDBFunctions(
  String catálogo,
  String patrón_esquema,
  String patrón_módulo,
  String patrón_nombre_función)
  throws java.sql.SQLException
```

Devuelve un objeto ResultSet que describe las funciones definidas por el usuario o las funciones incorporadas definidas en una fuente de datos. Las funciones para las que se devuelve información coinciden con el nombre de catálogo, el patrón de esquema, el patrón de módulo y el patrón de nombre de función especificados por los parámetros de entrada. ResultSet contiene las columnas siguientes:

Número de columna	Nombre de columna	Tipo de datos	Descripción
1	FUNCTION_CAT	Serie	Nombre del catálogo. El valor es nulo si esta función no tiene catálogos.
2	FUNCTION_SCHEM	Serie	Nombre del esquema que contiene FUNCTION_NAME. Este valor puede ser nulo.
3	FUNCTION_NAME	Serie	Nombre de la función.
4	REMARKS	Serie	Información descriptiva sobre el parámetro o nulo.
5	FUNCTION_TYPE	short	Tipo de función. Los valores posibles son: • functionResultUnknown (0): no se sabe si se devuelve un valor o una tabla. • functionNoTable (1): no se devuelve una tabla. • functionReturnsTable (2): se devuelve una tabla.
6	SPECIFIC_NAME	Serie	Nombre que identifica de forma exclusiva a la función dentro de su esquema.
7	FUNCTION_MODULE	Serie	Nombre del módulo que contiene la función. Este valor puede ser nulo.

Descripciones de parámetros:

catálogo

Una serie vacía o '%'.

schemaName

Especifica el nombre de esquema (calificador) de los objetos para los que se va a devolver información.

El valor puede identificar un nombre de esquema único o más de uno, incluyendo el carácter de porcentaje de coincidencia de patrones (%).

patrón_módulo
Especifica el nombre del módulo para el que se va a devolver información de parámetro.

El valor puede identificar un nombre de módulo único o más de uno, incluyendo el carácter de porcentaje de coincidencia de patrones (%).

patrón_nombre_función
Especifica el nombre de la función para la que se va a devolver información de parámetro.

El valor puede identificar un nombre de esquema único, o puede identificar más de un nombre de función incluyendo el carácter de porcentaje de coincidencia de patrones (%).

Este método solamente se aplica a conexiones con fuentes de datos de DB2 Database para Linux, UNIX y Windows 9.7 o posteriores. Se emite una SqlFeatureNotSupportedException si la fuente de datos es otra fuente de datos distinta.

getDBProcedureColumns
Formato:

```
public java.sql.ResultSet getDBProcedureColumns(
  String catálogo,
  String patrón_esquema,
  String patrón_módulo,
  String patrón_nombre_procedimiento,
  String patrón_nombre_columna)
  throws java.sql.SQLException
```

Devuelve un objeto ResultSet que describe los parámetros y valores de retorno para los procedimientos almacenados definidos en una fuente de datos. Los valores de retorno y los parámetros de procedimiento almacenado para los que se devuelve información coinciden con el nombre de catálogo, el patrón de esquema, el patrón de módulo, el patrón de nombre de procedimiento y el patrón de nombre de columna especificados por los parámetros de entrada. ResultSet contiene las columnas siguientes:

Número de columna	Nombre de columna	Tipo de datos	Descripción
1	PROCEDURE_CAT	Serie	Nombre del catálogo. El valor es nulo si este procedimiento no tiene catálogos.
2	PROCEDURE_SCHEM	Serie	Nombre del esquema que contiene PROCEDURE_NAME. Este valor puede ser nulo.
3	PROCEDURE_NAME	Serie	Nombre del procedimiento.
4	COLUMN_NAME	Serie	Nombre del parámetro de procedimiento.
5	COLUMN_TYPE	Short	Tipo de parámetro: • procedureColumnUnknown (0): desconocido • procedureColumnIn (1): entrada • procedureColumnInOut (2): entrada o salida • procedureColumnResult (3): columna de conjunto de resultados • procedureColumnOut (4): salida • procedureColumnReturn (5): valor devuelto
6	DATA_TYPE	int	Tipo de datos de SQL del parámetro identificado mediante COLUMN_NAME.

Número de columna	Nombre de columna	Tipo de datos	Descripción
7	TYPE_NAME	Serie	Serie de caracteres que representa el nombre del tipo de datos correspondiente a DATA_TYPE.
8	PRECISION	int	Precisión de un parámetro numérico exacto.
9	LENGTH	int	Longitud del parámetro.
10	SCALE	short	Escala de un parámetro numérico exacto.
11	RADIX	short	10, 2 o nulo: • Si DATA_TYPE es un tipo de datos numérico aproximado, este parámetro contiene el valor 2. • Si DATA_TYPE es un tipo de datos numérico exacto, este parámetro contiene el valor 10. • Para los tipos de datos en los que no es aplicable la base, este valor es nulo.
12	NULLABLE	short	Especifica si el parámetro puede contener valores NULL: **procedureNoNulls (0)** El parámetro no puede contener valores NULL. **procedureNullable (1)** El parámetro puede contener valores NULL. **procedureNullableUnknown (2)** No se sabe si el parámetro puede ser nulo.
13	REMARKS	Serie	Información descriptiva sobre el parámetro o nulo.
14	COLUMN_DEF	Serie	Valor por omisión para el parámetro.
15	SQL_DATA_TYPE	int	Valor del tipo de datos de SQL tal y como aparece en el campo SQL_DESC_TYPE del descriptor.
16	SQL_DATETIME_SUB	int	Código de subtipo para los tipos de datos de fecha y hora.
17	CHAR_OCTET_LENGTH	int	Para los tipos de datos de carácter, el número máximo de bytes en el parámetro. Para el tipo XML, se devuelve cero. Para el resto de los tipos de datos, este valor es nulo.
18	ORDINAL_POSITION	int	Posición del parámetro, empezando en 1. Este valor es 0 si COLUMN_TYPE es 5 (valor de retorno).
19	IS_NULLABLE	Serie	Contiene la serie 'NO' si se sabe que el parámetro no tiene posibilidad de ser nulo, 'YES' si el parámetro puede ser nulo o una serie vacía si no se sabe.
20	SPECIFIC_NAME	Serie	Nombre que identifica de forma exclusiva al procedimiento dentro de su esquema.
21	PROCEDURE_MODULE	Serie	Nombre del módulo que contiene el procedimiento. Este valor puede ser nulo.

Descripciones de parámetros:

catálogo
Una serie vacía o '%'.

schemaName
Especifica el nombre de esquema (calificador) de los objetos para los que se va a devolver información.

El valor puede identificar un nombre de esquema único o más de uno, incluyendo el carácter de porcentaje de coincidencia de patrones (%).

patrón_módulo
Especifica el nombre del módulo para el que se va a devolver información de parámetro.

El valor puede identificar un nombre de módulo único o más de uno, incluyendo el carácter de porcentaje de coincidencia de patrones (%).

procedureNamePattern
Especifica el nombre del procedimiento para el que se va a devolver información de parámetro.

El valor puede identificar un nombre de esquema único, o puede identificar más de un nombre de procedimiento incluyendo el carácter de porcentaje de coincidencia de patrones (%).

patrón_nombre_columna
Especifica los parámetros para los que se va a devolver información de parámetro.

El valor puede identificar un nombre de esquema único, o puede identificar más de un nombre de parámetro incluyendo el carácter de porcentaje de coincidencia de patrones (%).

Este método solamente se aplica a conexiones con fuentes de datos de DB2 Database para Linux, UNIX y Windows 9.7 o posteriores. Se emite una SqlFeatureNotSupportedException si la fuente de datos es otra fuente de datos distinta.

getDBProcedures
Formato:

```
public java.sql.ResultSet getDBProcedures(
  String catálogo,
  String patrón_esquema,
  String patrón_módulo,
  String patrón_nombre_procedimiento)
  throws java.sql.SQLException
```

Devuelve un objeto ResultSet que describe los procedimientos almacenados definidos en una fuente de datos. Los procedimientos almacenados para los que se devuelve información coinciden con el nombre de catálogo, el patrón de esquema, el patrón de módulo y el patrón de nombre de procedimiento especificados por los parámetros de entrada. ResultSet contiene las columnas siguientes:

Número de columna	Nombre de columna	Tipo de datos	Descripción
1	PROCEDURE_CAT	Serie	Nombre del catálogo. El valor es nulo si este procedimiento no tiene catálogos.
2	PROCEDURE_SCHEM	Serie	Nombre del esquema que contiene PROCEDURE_NAME. Este valor puede ser nulo.
3	PROCEDURE_NAME	Serie	Nombre del procedimiento.
4			Reservado.
5			Reservado.
6			Reservado.
7	REMARKS	Serie	Información descriptiva sobre el parámetro o nulo.

Número de columna	Nombre de columna	Tipo de datos	Descripción
8	PROCEDURE_TYPE	short	Tipo de procedimiento. Los valores posibles son: • procedureResultUnknown (0): no se sabe si se devuelve un valor. • procedureNoResult (1): no se devuelve ningún valor. • procedureReturnsResult (2): se devuelve un valor.
9	SPECIFIC_NAME	Serie	Nombre que identifica de forma exclusiva al procedimiento dentro de su esquema.
10	PROCEDURE_MODULE	Serie	Nombre del módulo que contiene el procedimiento. Este valor puede ser nulo.

Descripciones de parámetros:

catálogo
Una serie vacía o '%'.

schemaName
Especifica el nombre de esquema (calificador) de los objetos para los que se va a devolver información.

El valor puede identificar un nombre de esquema único o más de uno, incluyendo el carácter de porcentaje de coincidencia de patrones (%).

patrón_módulo
Especifica el nombre del módulo para el que se va a devolver información de parámetro.

El valor puede identificar un nombre de módulo único o más de uno, incluyendo el carácter de porcentaje de coincidencia de patrones (%).

procedureNamePattern
Especifica el nombre del procedimiento para el que se va a devolver información de parámetro.

El valor puede identificar un nombre de esquema único, o puede identificar más de un nombre de procedimiento incluyendo el carácter de porcentaje de coincidencia de patrones (%).

Este método solamente se aplica a conexiones con fuentes de datos de DB2 Database para Linux, UNIX y Windows 9.7 o posteriores. Se emite una SqlFeatureNotSupportedException si la fuente de datos es otra fuente de datos distinta.

getDBUDTs
Formato:

```
public java.sql.ResultSet getDBUDTs(
  String catálogo,
  String patrón_esquema,
  String patrón_módulo,
  String patrón_nombre_tipo,
  int[] tipos)
  throws java.sql.SQLException
```

Devuelve un objeto ResultSet que describe los tipos definidos por el usuario definidos en una fuente de datos. Los tipos definidos por el usuario para los que se devuelve información coinciden con el nombre de catálogo, el patrón de

esquema, el patrón de módulo, el patrón de nombre de tipo y los tipos especificados por los parámetros de entrada. ResultSet contiene las columnas siguientes:

Número de columna	Nombre de columna	Tipo de datos	Descripción
1	TYPE_CAT	Serie	Nombre del catálogo. El valor es nulo si este tipo definido por el usuario no tiene catálogos.
2	TYPE_SCHEM	Serie	Nombre del esquema que contiene TYPE_NAME. Este valor puede ser nulo.
3	TYPE_NAME	Serie	Nombre del tipo definido por el usuario.
4	CLASS_NAME	Serie	Nombre de clase de Java del tipo definido por el usuario.
5	DATA_TYPE	int	Uno de los valores de tipo siguientes que se define en java.sql.Types: • JAVA_OBJECT (2000) • DISTINCT (2001) • STRUCT (2002)
6	REMARKS	Serie	Información descriptiva sobre el tipo definido por el usuario o nulo.
7	BASE_TYPE	short	Si DATA_TYPE es DISTINCT, o DATA_TYPE es STRUCT y REFERENCE_GENERATION es USER_DEFINED, código de tipo de java.sql.Types del tipo de datos en el que se basa el tipo definido por el usuario. De lo contrario, este valor es nulo.
8	TYPE_MODULE	Serie	Nombre del módulo que contiene el tipo definido por el usuario. Este valor puede ser nulo.

Descripciones de parámetros:

catálogo
Una serie vacía o '%'.

schemaName
Especifica el nombre de esquema (calificador) de los objetos para los que se va a devolver información.

El valor puede identificar un nombre de esquema único o más de uno, incluyendo el carácter de porcentaje de coincidencia de patrones (%).

patrón_módulo
Especifica el nombre del módulo para el que se va a devolver información de parámetro.

El valor puede identificar un nombre de módulo único o más de uno, incluyendo el carácter de porcentaje de coincidencia de patrones (%).

patrón_nombre_tipo
Especifica el nombre del tipo definido por el usuario para el que se va a devolver información de parámetro.

El valor puede identificar un nombre de tipo único, o puede identificar más de un nombre de procedimiento incluyendo el carácter de porcentaje de coincidencia de patrones (%).

tipos
Especifica los tipos definidos por el usuario para los que se va a devolver información de parámetro. Cada entrada de la matriz puede contener uno de los valores siguientes:

- JAVA_OBJECT (2000)
- DISTINCT (2001)
- STRUCT (2002)

Si la matriz es nula, se devuelve información sobre todos los tipos.

Este método solamente se aplica a conexiones con fuentes de datos de DB2 Database para Linux, UNIX y Windows 9.7 o posteriores. Se emite una SqlFeatureNotSupportedException si la fuente de datos es otra fuente de datos distinta.

isIDSDatabaseAnsiCompliant

Formato:

```
public boolean isIDSDatabaseAnsiCompliant();
```

Devuelve `true` si la base de datos activa actual de IBM Informix es compatible con ANSI. En otro caso, devuelve false.

Una base de datos compatible con ANSI es una base de datos que se creó con la opción WITH LOG MODE ANSI.

Este método solamente es aplicable a conexiones con fuentes de datos IBM Informix. Se emite una excepción de SQL (SQLException) si la fuente de datos no es una fuente de datos IBM Informix.

isIDSDatabaseLogging

Formato:

```
public boolean isIDSDatabaseLogging();
```

Devuelve `true` si la base de datos IBM Informix activa actual es compatible con la anotación cronológica. En otro caso, devuelve false.

Una base de datos IBM Informix compatible con la anotación cronológica es una base de datos que se creó con la opción WITH LOG MODE ANSI, WITH BUFFERED LOG o WITH LOG.

Este método solamente es aplicable a conexiones con fuentes de datos IBM Informix. Se emite una excepción de SQL (SQLException) si la fuente de datos no es una fuente de datos IBM Informix.

isResetRequiredForDB2eWLM

Formato:

```
public boolean isResetRequiredForDB2eWLM();
```

Devuelve `true` si el servidor de bases de datos de destino necesita una reutilización limpia para trabajar con eWLM. En otro caso, devuelve false.

supportsDB2ProgressiveStreaming

Formato:

```
public boolean supportsDB2ProgressiveStreaming();
```

Devuelve `true` si la fuente de datos de destino es compatible con la modalidad continua progresiva. En otro caso, devuelve false.

Interfaz DB2Diagnosable

La interfaz com.ibm.db2.jcc.DB2Diagnosable proporciona un mecanismo para obtener información de diagnóstico de DB2 a partir de una SQLException.

Métodos DB2Diagnosable

Los métodos siguientes se definen únicamente para IBM Data Server Driver para JDBC y SQLJ.

getSqlca
Formato:

```
public DB2Sqlca getSqlca();
```

Devuelve un objeto DB2Sqlca a partir de una excepción java.sql.Exception que se genera al utilizar IBM Data Server Driver para JDBC y SQLJ.

getThrowable
Formato:

```
public Throwable getThrowable();
```

Devuelve un objeto java.lang.Throwable a partir de una excepción java.sql.Exception que se produce al utilizar IBM Data Server Driver para JDBC y SQLJ.

printTrace
Formato:

```
static public void printTrace(java.io.PrintWriter printWriter,
  String header);
```

Imprime información de diagnóstico después de emitirse una excepción java.sql.Exception al utilizar IBM Data Server Driver para JDBC y SQLJ.

Descripciones de parámetros:

printWriter
Es el destino de la información de diagnóstico.

header
Información definida por el usuario que se imprime al comienzo de la salida.

Clase DB2DataSource

La clase com.ibm.db2.jcc.DB2DataSource amplia la clase DB2BaseDataSource e implementa las interfaces javax.sql.DataSource, java.io.Serializable y javax.naming.Referenceable.

Métodos DB2DataSource

Los métodos siguientes se definen únicamente para IBM Data Server Driver para JDBC y SQLJ.

setSpecialRegisters
Formato:

```
public void setSpecialRegisters(java.util.Properties propiedades)
  throws java.sql.SQLException
```

Para cada par de clave y valor del objeto java.util.Properties, establece el registro especial del servidor de datos que la clave especifica para el valor correspondiente.

Este método no se aplica a las conexiones con servidores de datos IBM Informix.

Descripción de parámetros:

properties
Un objeto java.util.Properties que contiene pares de clave y valor, en el que cada clave es el nombre de un registro especial y cada valor es el valor en el que desea establecer ese registro especial. Por ejemplo, imaginemos que ds es un objeto DataSource anteriormente definido. Establezca los valores de las propiedades de la forma siguiente.

```
Properties prop = new Properties();
prop.add ("CURRENT SCHEMA", "SYSPROC");
prop.add ("CURRENT PACKAGESET", "PRODUCTION");
((com.ibm.db2.jcc.DB2BaseDataSource) ds).setSpecialRegisters(prop);
```

Clase DB2Driver

La clase com.ibm.db2.jcc.DB2Driver amplía la interfaz java.sql.Driver.

Métodos DB2Driver

Los métodos siguientes se definen únicamente para IBM Data Server Driver para JDBC y SQLJ.

changeDB2Password
Formato:

```
public static void changeDB2Password (String url,
 String userid,
 String oldPassword,
 String newPassword)
 throws java.sql.SQLException
```

Cambia la contraseña para acceder a un servidor de datos especificada por el parámetro url, para el usuario que se especifica mediante el parámetro userid. Este método puede cambiar una contraseña caducada o no caducado.

changeDB2Password se soporta para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 únicamente.

changeDB2Password no se soporta para las conexiones con IBM Informix.

Descripciones de parámetros:

url
El URL para el servidor de datos para el que se está cambiando la contraseña de un usuario. El valor *url* utiliza la sintaxis de un URL para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4.

userid
El usuario cuya contraseña se está cambiando.

oldPassword
La contraseña original del usuario.

newPassword
La contraseña nueva del usuario.

Clase DB2ExceptionFormatter

La clase com.ibm.db2.jcc.DB2ExceptionFormatter contiene métodos para imprimir información de diagnóstico en una corriente de datos.

Métodos DB2ExceptionFormatter

Los métodos siguientes se definen únicamente para IBM Data Server Driver para JDBC y SQLJ.

printTrace

Formatos:

```
static public void printTrace(java.sql.SQLException sqlException,
  java.io.PrintWriter printWriter, String header)

static public void printTrace(DB2Sqlca sqlca,
  java.io.PrintWriter printWriter, String header)

static public void printTrace(java.lang.Throwable throwable,
  java.io.PrintWriter printWriter, String header)
```

Imprime información de diagnóstico después de emitirse una excepción.

Descripciones de parámetros:

sqlException|sqlca|throwable
Excepción que se emitió durante una operación anterior de JDBC o Java.

printWriter
Es el destino de la información de diagnóstico.

header
Información definida por el usuario que se imprime al comienzo de la salida.

Clase DB2FileReference

La clase com.ibm.db2.jcc.DB2FileReference es una clase abstracta que define métodos que admiten la inserción de datos en tablas a partir de variables de referencia de archivo. Esta clase sólo se aplica a IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 para DB2 para z/OS, Versión 9 o posterior.

Campos DB2FileReference

Las siguientes constantes definen tipos de códigos solamente para IBM Data Server Driver para JDBC y SQLJ.

public static final short MAX_FILE_NAME_LENGTH = 255
Duración máxima del nombre de archivo para una variable de referencia de archivo.

Métodos DB2FileReference

Los métodos siguientes se definen únicamente para IBM Data Server Driver para JDBC y SQLJ.

getDriverType

Formato:

```
public int getDriverType()
```

Devuelve el tipo de datos del servidor de la variable de referencia de archivo. Este tipo es uno de los valores en com.ibm.db2.jcc.DB2Types.

getFileEncoding

Formato:

```
public String getFileEncoding()
```

Devuelve la codificación de los datos en el archivo para un objeto DB2FileReference.

getFileName

Formato:

```
public String getFileName()
```

Devuelve el nombre de archivo para un objeto DB2FileReference.

getFileCcsid

Formato:

```
public int getFileCcsid()
```

Devuelve el CCSID de los datos en el archivo para un objeto DB2FileReference.

setFileName

Formato:

```
public String setFileName(String fileName)
  throws java.sql.SQLException
```

Define el nombre de archivo en un objeto DB2FileReference.

Descripciones de parámetros:

fileName
Nombre del archivo de entrada para la variable de referencia de archivo. Se debe especificar el nombre en un archivo HFS existente.

Clase DB2JCCPlugin

La clase com.ibm.db2.jcc.DB2JCCPlugin es una clase abstracta que define métodos que se pueden implementar para proporcionar soporte de plugin de DB2 Database para Linux, UNIX y Windows. Esta clase solo atañe al DB2 Database para Linux, UNIX y Windows.

Métodos DB2JCCPlugin

Los métodos siguientes se definen únicamente para IBM Data Server Driver para JDBC y SQLJ.

getTicket

Formato:

```
public abstract byte[] getTicket(String user,
  String password,
  byte[] returnedToken)
  throws org.ietf.jgss.GSSException
```

Recupera un certificado Kerberos para un usuario.

Descripciones de parámetros:

user
ID de usuario para el que debe recuperarse el certificado Kerberos.

contraseña
Contraseña de *user*.

returnedToken

Interfaz DB2ParameterMetaData

La interfaz com.ibm.db2.jcc.DB2ParameterMetaData amplía la interfaz java.sql.ParameterMetaData.

Métodos DB2ParameterMetaData

Los métodos siguientes se definen únicamente para IBM Data Server Driver para JDBC y SQLJ.

getParameterMarkerNames

Formato:

```
public String[] getParameterMarkerNames()
  throws java.sql.SQLException
```

Devuelve una lista de nombres de marcador de parámetro que se utilizan en una sentencia de SQL.

Este método devuelve un valor nulo si la propiedad enableNamedParameterMarkers se define como DB2BaseDataSource.NOT_SET o DB2BaseDataSource.NO, o bien si no hay marcadores de parámetro con nombre en la sentencia de SQL.

getProcedureParameterName

Formato:

```
public String getProcedureParameterName(int param)
  throws java.sql.SQLException
```

Devuelve el nombre en la sentencia CREATE PROCEDURE de un parámetro en una sentencia CALL de SQL. Si el parámetro no tiene nombre en la sentencia CREATE PROCEDURE, se devuelve la posición ordinal del parámetro en la sentencia CREATE PROCEDURE.

Descripciones de parámetros:

param
La posición ordinal del parámetro en la sentencia CALL.

Este método solamente se aplica a conexiones con servidores de datos de DB2 Database para Linux, UNIX y Windows 9.7 o posteriores.

Clase DB2PooledConnection

La clase com.ibm.db2.jcc.DB2PooledConnection ofrece métodos que puede utilizar un servidor de aplicaciones para cambiar de usuarios en una conexión fiable preexistente.

Las conexiones fiables están soportadas para:

- IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 para:
  - DB2 Database para Linux, UNIX y Windows Versión 9.5 o versiones posteriores
  - DB2 para z/OS Versión 9.1 o versiones posteriores
  - IBM Informix Versión 11.70 o versiones posteriores
- IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS Versión 9.1 o versiones posteriores

Métodos DB2PooledConnection

Los métodos siguientes se definen únicamente para IBM Data Server Driver para JDBC y SQLJ.

getConnection (reutilización no fiable o fiable sin reautentificación)

Formato:

```
public DB2Connection getConnection()
  throws java.sql.SQLException
```

Este método sirve para la *dirty reuse* de una conexión. Esto significa que el estado de la conexión no se restablece cuando se reutiliza el objeto desde la agrupación. Los valores de propiedad y los valores de registro especiales siguen vigentes a menos que los alteren temporalmente las propiedades pasadas. No se suprimen las tablas temporales globales. Las propiedades que no se especifiquen no se volverán a inicializar. Todas las propiedades transitorias estándar de JDBC, como el nivel de aislamiento, la modalidad de confirmación automático y la modalidad de sólo lectura se restablecen a sus valores por omisión de JDBC. Ciertas propiedades, como por ejemplo, `user`, `password`, `databaseName`, `serverName`, `portNumber`, `planName` y `pkList` permanecerán sin modificar.

getDB2Connection (reutilización fiable)

Formatos:

```
public DB2Connection getDB2Connection(byte[] cookie,
  String user,
  String password,
  String userRegistry,
  byte[] userSecToken,
  String originalUser,
  java.util.Properties properties)
  throws java.sql.SQLException
public Connection getDB2Connection(byte[] cookie,
  org.ietf.GSSCredential gssCredential,
  String usernameRegistry,
  byte[] userSecToken,
  String originalUser,
  java.util.Properties properties)
  throws java.sql.SQLException
```

Cambia el usuario que está asociado a una conexión fiable sin autentificación.

El segundo formato de getDB2Connection sólo está soportado para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4.

Descripciones de parámetros:

cookie
Cookie exclusivo creado por el controlador JDBC para la instancia de Connection. Sólo el servidor de aplicaciones y el controlador JDBC subyacente que estableció la conexión fiable inicial conocen la cookie. El servidor de aplicaciones pasa la cookie creada por el controlador en el momento de la creación de la instancia de la conexión agrupada. El controlador JDBC comprueba que la cookie proporcionada coincida con la cookie de la conexión física fiable subyacente para garantizar que la petición se ha creado a partir del servidor de aplicaciones que ha establecido la conexión física fiable. Si las cookies difieren, la conexión puede quedar disponible, con propiedades diferentes, para que un usuario nuevo la utilice inmediatamente.

user
La identidad cliente utilizada por la fuente de datos para establecer el ID de autorización para el servidor de bases de datos. Si el servidor de aplicaciones no autentificó al usuario, el primero debe pasar una identidad de usuario que represente a un usuario sin autentificar.

contraseña
Contraseña de *user*.

gssCredential
Si la fuente de datos utiliza la seguridad Kerberos, especifica una credencial delegada que se pasa desde otro principal.

userNameRegistry
Nombre que identifica un servicio de correlación que correlaciona un ID de usuario de estación de trabajo con un ID de z/OS RACF. Un ejemplo de servicio de correlación es la correlación de identidades empresariales (EIM) de servicios de seguridad integrados. El servicio de correlación se define mediante un plugin. Los proveedores de plugins definen los valores válidos de *userNameRegistry*. Si *userNameRegistry* es nulo, la conexión no utiliza un servicio de correlación.

userSecToken
Símbolos de seguridad del cliente. Este valor se rastrea como parte de datos contables de DB2 para z/OS. El contenido de *userSecToken* es descrito por el servidor de aplicaciones y es especificado por la fuente de datos como un símbolo de seguridad del servidor de aplicaciones.

originalUser
La identidad cliente que envía la petición original al servidor de aplicaciones. *originalUser* se incluye en los datos contables de DB2 para z/OS en calidad de ID de usuario original que fue utilizado por el servidor de aplicaciones.

propiedades
Propiedades de la conexión reutilizada. Estas propiedades alteran temporalmente las propiedades ya definidas en la instancia de DB2PooledConnection.

getDB2Connection (reutilización no fiable con reautentificación)
Formatos:

```
public DB2Connection getDB2Connection(
  String user,
  String password,
  java.util.Properties properties)
  throws java.sql.SQLException
public DB2Connection getDB2Connection(org.ietf.jgss.GSSCredential gssCredential,
  java.util.Properties properties)
  throws java.sql.SQLException
```

Cambia el usuario que está asociado a una conexión no fiable con autentificación.

El primer formato de getDB2Connection ofrece un ID de usuario y una contraseña. El segundo formato de getDB2Connection es para conexiones que utilizan la seguridad Kerberos.

Descripciones de parámetros:

user
El ID de usuario utilizado por la fuente de datos para establecer el ID de autorización para el servidor de bases de datos.

contraseña
Contraseña de *user*.

propiedades
Propiedades de la conexión reutilizada. Estas propiedades alteran temporalmente las propiedades ya definidas en la instancia de DB2PooledConnection.

getDB2Connection (reutilización no fiable o fiable sin reautentificación)

Formatos:

```
public java.sql.Connection getDB2Connection(
  java.util.Properties properties)
  throws java.sql.SQLException
```

Reutiliza una conexión no fiable, sin reautentificación.

Este método sirve para la *dirty reuse* de una conexión. Esto significa que el estado de la conexión no se restablece cuando se reutiliza el objeto desde la agrupación. Los valores de propiedad y los valores de registro especiales siguen vigentes a menos que los alteren temporalmente las propiedades pasadas. No se suprimen las tablas temporales globales. Las propiedades que no se especifiquen no se volverán a inicializar. Todas las propiedades transitorias estándar de JDBC, como el nivel de aislamiento, la modalidad de confirmación automático y la modalidad de sólo lectura se restablecen a sus valores por omisión de JDBC. Ciertas propiedades, como por ejemplo, `user`, `password`, `databaseName`, `serverName`, `portNumber`, `planName` y `pkList` permanecerán sin modificar.

Descripciones de parámetros:

propiedades
Propiedades de la conexión reutilizada. Estas propiedades alteran temporalmente las propiedades ya definidas en la instancia de DB2PooledConnection.

Clase DB2PoolMonitor

La clase com.ibm.db2.jcc.DB2PoolMonitor proporciona métodos para la supervisión de la agrupación de objetos de transporte global que se utiliza para el concentrador de conexiones y el equilibrado de carga de trabajo Sysplex.

Campos de DB2PoolMonitor

Los campos siguientes se definen solamente para IBM Data Server Driver para JDBC y SQLJ.

public static final int TRANSPORT_OBJECT = 1
Este valor es un parámetro para el método DB2PoolMonitor.getPoolMonitor.

Métodos DB2PoolMonitor

Los métodos siguientes se definen únicamente para IBM Data Server Driver para JDBC y SQLJ.

agedOutObjectCount

Formato:

```
public abstract int agedOutObjectCount()
```

Recupera el número de objetos que han excedido el tiempo de inactividad especificado por db2.jcc.maxTransportObjectIdleTime y que no se han suprimido de la agrupación.

createdObjectCount

Formato:

```
public abstract int createdObjectCount()
```

Recupera el número de objetos que IBM Data Server Driver para JDBC y SQLJ ha creado desde la creación de la agrupación.

getMonitorVersion

Formato:

```
public int getMonitorVersion()
```

Recupera la versión de la clase DB2PoolMonitor que se entrega con IBM Data Server Driver para JDBC y SQLJ.

getPoolMonitor

Formato:

```
public static DB2PoolMonitor getPoolMonitor(int tipoMonitor)
```

Recupera una instancia de la clase DB2PoolMonitor.

Descripciones de parámetros:

tipoMonitor
El tipo de monitor. Este valor debe ser DB2PoolMonitor.TRANSPORT_OBJECT.

heavyWeightReusedObjectCount

Formato:

```
public abstract int heavyWeightReusedObjectCount()
```

Recupera el número de la agrupación que se han vuelto a utilizar.

lightWeightReusedObjectCount

Formato:

```
public abstract int lightWeightReusedObjectCount()
```

Recupera el número de objetos que se han vuelto a utilizar pero que no se encontraban en la agrupación. Esto puede ocurrir si un objeto Connection libera un objeto de transporte en el límite de una transacción. Si el objeto Connection necesita un objeto de transporte posteriormente y ningún otro objeto Connection ha utiliza el objeto de transporte original, el objeto Connection podrá utilizar el objeto de transporte.

longestBlockedRequestTime

Formato:

```
public abstract long longestBlockedRequestTime()
```

Recupera, en milisegundos, la mayor cantidad de tiempo que una petición estuvo bloqueada.

numberOfConnectionReleaseRefused

Formato:

```
public abstract int numberOfConnectionReleaseRefused()
```

Recupera el número de veces que se rechazó la liberación de una conexión.

numberOfRequestsBlocked

Formato:

```
public abstract int numberOfRequestsBlocked()
```

Recupera el número de peticiones que IBM Data Server Driver para JDBC y SQLJ ha realizado a la agrupación que la agrupación bloqueó debido a que ésta había llegado a su capacidad máxima. Es posible que una petición

bloqueada se ejecute correctamente si el objeto se devuelve a la agrupación antes de que se supere el tiempo especificado por db2.jcc.maxTransportObjectWaitTime y que se haya emitido una excepción.

numberOfRequestsBlockedDataSourceMax

Formato:

```
public abstract int numberOfRequestsBlockedDataSourceMax()
```

Recupera el número de peticiones que IBM Data Server Driver para JDBC y SQLJ ha realizado a la agrupación que la agrupación bloqueó debido a que ésta había llegado al máximo del objeto DataSource.

numberOfRequestsBlockedPoolMax

Formato:

```
public abstract int numberOfRequestsBlockedPoolMax()
```

Recupera el número de peticiones que IBM Data Server Driver para JDBC y SQLJ ha realizado a la agrupación que la agrupación bloqueó debido a que ésta había llegado al máximo de la agrupación.

removedObjectCount

Formato:

```
public abstract int removedObjectCount()
```

Recupera el número de objetos que se han suprimido de la agrupación desde que ésta fue creada.

shortestBlockedRequestTime

Formato:

```
public abstract long shortestBlockedRequestTime()
```

Recupera, en milisegundos, la menor cantidad de tiempo que una petición estuvo bloqueada.

successfullRequestsFromPool

Formato:

```
public abstract int successfullRequestsFromPool()
```

Recupera el número de peticiones satisfactorias que IBM Data Server Driver para JDBC y SQLJ ha realizado a la agrupación desde la creación de ésta. Una petición satisfactoria indica que la agrupación ha devuelto un objeto.

totalPoolObjects

Formato:

```
public abstract int totalPoolObjects()
```

Recupera el número de objetos que se encuentran actualmente en la agrupación.

totalRequestsToPool

Formato:

```
public abstract int totalRequestsToPool()
```

Recupera el número total de peticiones que IBM Data Server Driver para JDBC y SQLJ ha realizado a la agrupación desde la creación de ésta.

totalTimeBlocked

Formato:

```
public abstract long totalTimeBlocked()
```

Recupera el tiempo total en milisegundos que la agrupación ha empleado para bloquear las peticiones. Este tiempo puede ser mucho mayor que el tiempo de ejecución de la aplicación transcurrido si la aplicación utiliza varias hebras.

Interfaz DB2PreparedStatement

La interfaz com.ibm.db2.jcc.DB2PreparedStatement amplía las interfaces com.ibm.db2.jcc.DB2Statement y java.sql.PreparedStatement.

Campos de DB2PreparedStatement

Las constantes siguientes se definen solamente para IBM Data Server Driver para JDBC y SQLJ.

public static DBIndicatorDefault DB_PARAMETER_DEFAULT
Esta constante puede utilizarse con interfaces estándar, como PreparedStatement.setObject o ResultSet.updateObject, para indicar que el valor por omisión se asigna al parámetro asociado.

public static DBIndicatorUnassigned DB_PARAMETER_UNASSIGNED
Esta constante puede utilizarse con interfaces estándar, como PreparedStatement.setObject o ResultSet.updateObject, para indicar que el parámetro no está asignado.

Métodos DB2PreparedStatement

Los métodos siguientes se definen únicamente para IBM Data Server Driver para JDBC y SQLJ.

executeDB2QueryBatch
Formato:

```
public void executeDB2QueryBatch()
  throws java.sql.SQLException
```

Ejecuta un lote de sentencias que contiene consultas con parámetros.

Este método no se puede utilizar para conexiones con fuentes de datos IBM Informix.

getDBGeneratedKeys
Formato:

```
public java.sql.ResultSet[] getDBGeneratedKeys()
  throws java.sql.SQLException
```

Recupera claves generadas automáticamente que se crearon cuando se ejecutó un lote de sentencias INSERT. Cada objeto ResultSet devuelto contiene las claves generadas automáticamente correspondiente a una sentencia individual del lote.

getDBGeneratedKeys devuelve una matriz de longitud 0 en las condiciones siguientes:

- getDBGeneratedKeys se invoca fuera de secuencia. Por ejemplo, si getDBGeneratedKeys se invoca antes que executeBatch, se devuelve una matriz de longitud 0.
- La sentencia preparada (PreparedStatement) que se ejecuta en un lote no se ha creado utilizando uno de estos métodos:

```
Connection.prepareStatement(String sql, int[] autoGeneratedKeys)
Connection.prepareStatement(String sql, String[] autoGeneratedColumnNames)
Connection.prepareStatement(String sql, Statement.RETURN_GENERATED_KEYS)
```

Si getDBGeneratedKeys se ejecuta para una sentencia preparada (PreparedStatement) que se ha creado utilizando uno de los métodos indicados anteriormente, y la sentencia preparada (PreparedStatement) no está en un lote, se devuelve un solo conjunto de resultados (ResultSet).

getEstimateCost

Formato:

```
public int getEstimateCost()
  throws java.sql.SQLException
```

Devuelve el coste estimado de una sentencia de SQL del servidor de datos después de que el servidor de datos prepare la sentencia de forma dinámica correctamente. Este valor es el mismo que el cuarto elemento de la matriz sqlerrd de la SQLCA.

Si la propiedad deferPrepares está establecida en `true`, la llamada a getEstimateCost hace que el servidor de datos ejecute una operación de preparación dinámica.

Si la sentencia de SQL no se puede preparar, o si el servidor de datos no devuelve la información de coste estimado en el momento de la preparación, getEstimateCost devuelve -1.

getEstimateRowCount

Formato:

```
public int getEstimateRowCount()
  throws java.sql.SQLException
```

Devuelve el número de filas estimado de una sentencia de SQL del servidor de datos después de que el servidor de datos prepare la sentencia de forma dinámica correctamente. Este valor es el mismo que el tercer elemento de la matriz sqlerrd de la SQLCA.

Si la propiedad deferPrepares está establecida en `true`, la llamada a getEstimateRowCount hace que el servidor de datos ejecute una operación de preparación dinámica.

Si la sentencia de SQL no se puede preparar, o si el servidor de datos no devuelve la información de número de filas estimado en el momento de la preparación, getEstimateRowCount devuelve -1.

setDBTimestamp

Formato:

```
public void setDBTimestamp(int
Índice Parámetro,
  DBTimestamp IndicaciónFechaHora)
  throws java.sql.SQLException
```

Asigna un valor DBTimestamp a un parámetro.

Parámetros:

índiceParámetro
: Índice del marcador de parámetro al que se asigna un valor de variable de DBTimestamp.

IndicaciónFechaHora
: El valor DBTimestamp que se asigna al marcador de parámetro.

Este método no se puede utilizar para conexiones con fuentes de datos IBM Informix.

setJccArrayAtName

Formato:

```
public void setJccArrayAtName(String nombreMarcadorParámetro,
  java.sql.Array x)
  throws java.sql.SQLException
```

Asigna un valor java.sql.Array a un marcador de parámetro con nombre.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en `DB2BaseDataSource.YES` (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro al que se asigna un valor.

x El valor java.sql.Array que se asigna al marcador de parámetro con nombre.

setJccAsciiStreamAtName

Formatos:

Soportado por IBM Data Server Driver para JDBC y SQLJ Versión 3.57 y posteriores:

```
public void setJccAsciiStreamAtName(String nombreMarcadorParámetro,
  java.io.InputStream x, int longitud)
  throws java.sql.SQLException
```

Soportado por IBM Data Server Driver para JDBC y SQLJ Versión 4.7 y posteriores:

```
public void setJccAsciiStreamAtName(String nombreMarcadorParámetro,
  java.io.InputStream x)
  throws java.sql.SQLException
public void setJccAsciiStreamAtName(String nombreMarcadorParámetro,
  java.io.InputStream x, long longitud)
  throws java.sql.SQLException
```

Asigna un valor ASCII de java.io.InputStream a un marcador de parámetro con nombre.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en `DB2BaseDataSource.YES` (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro al que se asigna un valor.

x El valor ASCII java.io.InputStream que se asigna al marcador de parámetro.

longitud
Longitud en bytes del valor java.io.InputStream que se asigna al marcador de parámetro con nombre.

setJccBigDecimalAtName

Formato:

```
public void setJccBigDecimalAtName(String nombreMarcadorParámetro,
  java.math.BigDecimal x)
  throws java.sql.SQLException
```

Asigna un valor java.math.BigDecimal a un marcador de parámetro con nombre.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en DB2BaseDataSource.YES (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro al que se asigna un valor.

x El valor java.math.BigDecimal que se asigna al marcador de parámetro con nombre.

setJccBinaryStreamAtName

Formatos:

Soportado por IBM Data Server Driver para JDBC y SQLJ Versión 3.57 y posteriores:

```
public void setJccBinaryStreamAtName(String nombreMarcadorParámetro,
  java.io.InputStream x, int longitud)
  throws java.sql.SQLException
```

Soportado por IBM Data Server Driver para JDBC y SQLJ Versión 4.7 y posteriores:

```
public void setJccBinaryStreamAtName(String nombreMarcadorParámetro,
  java.io.InputStream x)
  throws java.sql.SQLException
public void setJccBinaryStreamAtName(String nombreMarcadorParámetro,
  java.io.InputStream x, long longitud)
  throws java.sql.SQLException
```

Asigna un valor binario de java.io.InputStream a un marcador de parámetro con nombre.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en DB2BaseDataSource.YES (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro al que se asigna un valor.

x El valor binario java.io.InputStream que se asigna al marcador de parámetro.

longitud
Número de bytes del valor java.io.InputStream que se asignan al marcador de parámetro con nombre.

setJccBlobAtName

Formatos:

Soportado por IBM Data Server Driver para JDBC y SQLJ Versión 3.57 y posteriores:

```
public void setJccBlobAtName(String nombreMarcadorParámetro,
  java.sql.Blob x)
  throws java.sql.SQLException
```

Soportado por IBM Data Server Driver para JDBC y SQLJ Versión 4.7 y posteriores:

```
public void setJccBlobAtName(String nombreMarcadorParámetro,
  java.io.InputStream x)
  throws java.sql.SQLException
public void setJccBlobAtName(String nombreMarcadorParámetro,
  java.io.InputStream x, long longitud)
  throws java.sql.SQLException
```

Asigna un valor BLOB a un marcador de parámetro con nombre.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en DB2BaseDataSource.YES (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro al que se asigna un valor.

x El valor java.sql.Blob o java.io.InputStream que se asigna al marcador de parámetro.

longitud
Número de bytes del valor java.io.InputStream que se asignan al marcador de parámetro con nombre.

setJccBooleanAtName

Formato:

```
public void setJccBooleanAtName(String nombreMarcadorParámetro,
  boolean x)
  throws java.sql.SQLException
```

Asigna un valor booleano a un marcador de parámetro con nombre.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en DB2BaseDataSource.YES (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro al que se asigna un valor.

x El valor booleano que se asigna al marcador de parámetro con nombre.

setJccByteAtName

Formato:

```
public void setJccByteAtName(String nombreMarcadorParámetro,
  byte x)
  throws java.sql.SQLException
```

Asigna un valor byte a un marcador de parámetro con nombre.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en DB2BaseDataSource.YES (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro al que se asigna un valor.

x El valor de byte que se asigna al marcador de parámetro con nombre.

setJccBytesAtName

Formato:

```
public void setJccBytesAtName(String nombreMarcadorParámetro,
  byte[] x)
  throws java.sql.SQLException
```

Asigna una matriz de valores de byte a un marcador de parámetro con nombre.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en DB2BaseDataSource.YES (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro al que se asigna un valor.

x La matriz de bytes que se asigna al marcador de parámetro con nombre.

setJccCharacterStreamAtName

Formatos:

Soportado por IBM Data Server Driver para JDBC y SQLJ Versión 3.57 y posteriores:

```
public void setJccCharacterStreamAtName(String nombreMarcadorParámetro,
  java.io.Reader x, int longitud)
  throws java.sql.SQLException
```

Soportado por IBM Data Server Driver para JDBC y SQLJ Versión 4.7 y posteriores:

```
public void setJccCharacterStreamAtName(String nombreMarcadorParámetro,
  java.io.Reader x)
  throws java.sql.SQLException
public void setJccCharacterStreamAtName(String nombreMarcadorParámetro,
  java.io.Reader x, long longitud)
  throws java.sql.SQLException
```

Asigna un valor Unicode de java.io.Reader a un marcador de parámetro con nombre.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en `DB2BaseDataSource.YES` (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro al que se asigna un valor.

x El valor Unicode java.io.Reader que se asigna al marcador de parámetro con nombre.

longitud
Número de caracteres del valor java.io.InputStream que se asignan al marcador de parámetro con nombre.

setJccClobAtName

Formatos:

Soportado por IBM Data Server Driver para JDBC y SQLJ Versión 3.57 y posteriores:

```
public void setJccClobAtName(String nombreMarcadorParámetro,
  java.sql.Clob x)
  throws java.sql.SQLException
```

Soportado por IBM Data Server Driver para JDBC y SQLJ Versión 4.7 y posteriores:

```
public void setJccClobAtName(String nombreMarcadorParámetro,
  java.io.Reader x)
  throws java.sql.SQLException
public void setJccClobAtName(String nombreMarcadorParámetro,
  java.io.Reader x, long longitud)
  throws java.sql.SQLException
```

Asigna un valor CLOB a un marcador de parámetro con nombre.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en DB2BaseDataSource.YES (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro al que se asigna un valor.

x El valor java.sql.Clob o java.io.Reader que se asigna al marcador de parámetro con nombre.

longitud
Número de bytes del valor java.io.InputStream que se asignan al marcador de parámetro con nombre.

setJccDateAtName

Formatos:

```
public void setJccDateAtName(String nombreMarcadorParámetro,
  java.sql.Date x)
  throws java.sql.SQLException
public void setJccDateAtName(String nombreMarcadorParámetro,
  java.sql.Date x,
  java.util.Calendar cal)
  throws java.sql.SQLException
```

Asigna un valor java.sql.Date a un marcador de parámetro con nombre.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en DB2BaseDataSource.YES (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro al que se asigna un valor.

x El valor java.sql.Date que se asigna al marcador de parámetro con nombre.

cal
El objeto java.util.Calendar que utiliza IBM Data Server Driver para JDBC y SQLJ para construir la fecha.

setJccDBTimestampAtName

Formato:

```
public void setJccDBTimestampAtName(String
nombreMarcadorParámetro,
  DBTimestamp IndicaciónFechaHora)
  throws java.sql.SQLException
```

Asigna un valor DBTimestamp a un marcador de parámetro con nombre.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en DB2BaseDataSource.YES (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro al que se asigna un valor de variable de DBTimestamp.

IndicaciónFechaHora
El valor DBTimestamp que se asigna al marcador de parámetro con nombre.

Este método no se puede utilizar para conexiones con fuentes de datos IBM Informix.

setJccDBDefaultAtName

Formatos:

```
public void setJccDBDefaultAtName(String nombreMarcadorParámetro)
  throws SQLException
```

Asigna el valor por omisión a un marcador de parámetro con nombre. La ejecución de setJccDBDefaultAtName produce los mismos resultados que utilizar el literal DEFAULT en la serie SQL, en lugar del nombre del marcador de parámetro.

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro al que se asigna un valor.

Este método no se puede utilizar para conexiones con fuentes de datos IBM Informix.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en DB2BaseDataSource.YES (1).

setJccDBUnassignedAtName

Formatos:

```
public void setJccDBUnassignedAtName(String nombreMarcadorParámetro)
  throws SQLException
```

No asigna un valor al parámetro con nombre especificado. La ejecución de setJccDBUnassignedAtName genera el mismo resultado que si el nombre de marcador de parámetro especificado no hubiera aparecido en la serie de SQL.

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro cuyo valor debe ser UNASSIGNED.

Este método no se puede utilizar para conexiones con fuentes de datos IBM Informix.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en DB2BaseDataSource.YES (1).

setJccDoubleAtName

Formato:

```
public void setJccDoubleAtName(String nombreMarcadorParámetro,
  double x)
  throws java.sql.SQLException
```

Asigna un valor de tipo double a un marcador de parámetro con nombre.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en DB2BaseDataSource.YES (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro al que se asigna un valor.

x El valor de tipo double que se asigna al marcador de parámetro.

setJccFloatAtName

Formato:

```
public void setJccFloatAtName(String nombreMarcadorParámetro,
  float x)
  throws java.sql.SQLException
```

Asigna un valor de tipo float a un marcador de parámetro con nombre.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en DB2BaseDataSource.YES (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro al que se asigna un valor.

x El valor de tipo float que se asigna al marcador de parámetro.

setJccIntAtName

Formato:

```
public void setJccIntAtName(String nombreMarcadorParámetro,
  int x)
  throws java.sql.SQLException
```

Asigna un valor de tipo int a un marcador de parámetro con nombre.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en DB2BaseDataSource.YES (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro al que se asigna un valor.

x El valor de tipo int que se asigna al marcador de parámetro.

setJccLongAtName

Formato:

```
public void setJccLongAtName(String nombreMarcadorParámetro,
  long x)
  throws java.sql.SQLException
```

Asigna un valor de tipo long a un marcador de parámetro con nombre.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en DB2BaseDataSource.YES (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro al que se asigna un valor.

x El valor de tipo long que se asigna al marcador de parámetro.

setJccNullAtName

Formato:

```
public void setJccNullAtName(String nombreMarcadorParámetro,
  int Tipojdbc)
  throws java.sql.SQLException
public void setJccNullAtName(String nombreMarcadorParámetro,
  int Tipojdbc,
  String typeName)
  throws java.sql.SQLException
```

Asigna el valor SQL NULL a un marcador de parámetro con nombre.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en DB2BaseDataSource.YES (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro al que se asigna un valor.

Tipojdbc
El código de tipo JDBC del valor NULL que se asigna al marcador de parámetro, tal y como se define en java.sql.Types.

typeName
Si *Tipojdbc* es java.sql.Types.DISTINCT o java.sql.Types.REF, el nombre totalmente calificado del tipo SQL definido por el usuario del valor NULL que se asigna al marcador de parámetro.

setJccObjectAtName

Formatos:

```
public void setJccObjectAtName(String nombreMarcadorParámetro,
  java.sql.Object x)
  throws java.sql.SQLException
public void setJccObjectAtName(String nombreMarcadorParámetro,
  java.sql.Object x,
  int TipoJdbcDestino)
  throws java.sql.SQLException
public void setJccObjectAtName(String nombreMarcadorParámetro,
  java.sql.Object x,
  int TipoJdbcDestino,
  int escala)
  throws java.sql.SQLException
```

Asigna un valor con el tipo java.lang.Object a un marcador de parámetro con nombre.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en `DB2BaseDataSource.YES` (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro al que se asigna un valor.

x El valor con tipo Object que se asigna al marcador de parámetro.

TipoJdbcDestino
El tipo de datos, tal y como se define en java.sql.Types, que se asigna al valor de entrada cuando se envía a la fuente de datos.

escala
Escala del valor que se asigna al marcador de parámetro. Este parámetro se aplica únicamente en los casos siguientes:

- Si *TipoJdbcDestino* es java.sql.Types.DECIMAL o java.sql.Types.NUMERIC, *escala* es el número de dígitos situados a la derecha de la coma decimal.
- Si *x* tiene el tipo java.io.InputStream o java.io.Reader, *escala* es la longitud de los datos del objeto Stream o Reader.

setJccShortAtName

Formato:

```
public void setJccShortAtName(String nombreMarcadorParámetro,
  short x)
  throws java.sql.SQLException
```

Asigna un valor de tipo short a un marcador de parámetro con nombre.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en `DB2BaseDataSource.YES` (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro al que se asigna un valor.

x El valor de tipo short que se asigna al marcador de parámetro.

setJccSQLXMLAtName

Formato:

```
public void setJccSQLXMLAtName(String nombreMarcadorParámetro,
  java.sql.SQLXML x)
  throws java.sql.SQLException
```

Asigna un valor de tipo java.sql.SQLXML a un marcador de parámetro con nombre.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en `DB2BaseDataSource.YES` (1).

Este método sólo se puede utilizar para conexiones con DB2 Database para Linux, UNIX y Windows Versión 9.1 o posterior, o bien DB2 para z/OS Versión 9 o posterior.

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro al que se asigna un valor.

x El valor de tipo java.sql.SQLXML que se asigna al marcador de parámetro.

setJccStringAtName

Formato:

```
public void setJccStringAtName(String nombreMarcadorParámetro,
  String x)
  throws java.sql.SQLException
```

Asigna un valor de tipo String a un marcador de parámetro con nombre.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en `DB2BaseDataSource.YES` (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro al que se asigna un valor.

x El valor de tipo String que se asigna al marcador de parámetro.

setJccTimeAtName

Formatos:

```
public void setJccTimeAtName(String nombreMarcadorParámetro,
  java.sql.Time x)
  throws java.sql.SQLException
public void setJccTimeAtName(String nombreMarcadorParámetro,
  java.sql.Time x,
  java.util.Calendar cal)
  throws java.sql.SQLException
```

Asigna un valor java.sql.Time a un marcador de parámetro con nombre.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en `DB2BaseDataSource.YES` (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro al que se asigna un valor.

x El valor java.sql.Time que se asigna al marcador de parámetro.

cal
El objeto java.util.Calendar que utiliza IBM Data Server Driver para JDBC y SQLJ para construir la hora.

setJccTimestampAtName

Formatos:

```
public void setJccTimestampAtName(String nombreMarcadorParámetro,
  java.sql.Timestamp x)
  throws java.sql.SQLException
public void setJccTimestampAtName(String nombreMarcadorParámetro,
  java.sql.Timestamp x,
  java.util.Calendar cal)
  throws java.sql.SQLException
```

Asigna un valor java.sql.Timestamp a un marcador de parámetro con nombre.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en `DB2BaseDataSource.YES` (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro al que se asigna un valor.

x El valor java.sql.Timestamp que se asigna al marcador de parámetro.

cal
El objeto java.util.Calendar que utiliza IBM Data Server Driver para JDBC y SQLJ para construir la indicación de fecha y hora.

setJccUnicodeStreamAtName

Formato:

```
public void setJccUnicodeStreamAtName(String nombreMarcadorParámetro,
  java.io.InputStream x, int longitud)
  throws java.sql.SQLException
```

Asigna un valor Unicode de java.io.InputStream a un marcador de parámetro con nombre.

Solamente se podrá llamar a este método si la propiedad enableNamedParameterMarkers está establecida en `DB2BaseDataSource.YES` (1).

Parámetros:

nombreMarcadorParámetro
Nombre del marcador de parámetro al que se asigna un valor.

x El valor Unicode java.io.InputStream que se asigna al marcador de parámetro.

longitud
Número de bytes del valor java.io.InputStream que se asignan al marcador de parámetro.

setDBDefault

Formatos:

```
public void setDBDefault(int índiceParámetro)
  throws SQLException
```

Asigna el valor por omisión al parámetro especificado. La ejecución de setDBDefault produce los mismos resultados que utilizar el literal DEFAULT en la serie de SQL, en lugar del parámetro.

Parámetros:

índiceParámetro
Número del parámetro cuyo valor se está actualizando.

Este método no se puede utilizar para conexiones con fuentes de datos IBM Informix.

setDBUnassigned

Formatos:

```
public void setDBUnassigned(int índiceParámetro)
  throws SQLException
```

No asigna un valor al parámetro especificado. La ejecución de setDBUnassigned genera el mismo resultado que si el parámetro especificado no hubiera aparecido en la serie de SQL.

Parámetros:

índiceParámetro
Número del parámetro cuyo valor debe ser UNASSIGNED.

Este método no se puede utilizar para conexiones con fuentes de datos IBM Informix.

Interfaz DB2ResultSet

La interfaz com.ibm.db2.jcc.DB2ResultSet se utiliza para crear objetos de los cuales se puede obtener información de consulta para IBM Data Server Driver para JDBC y SQLJ.

DB2ResultSet implementa la interfaz java.sql.Wrapper.

Campos de DB2ResultSet

Los campos siguientes se definen solamente para IBM Data Server Driver para JDBC y SQLJ.

Las constantes de enteros de la tabla siguiente se utilizan en la información del descriptor de columna que getDBRowDescriptor devuelve. Estas constantes contienen información acerca de los valores de columna que getDBRowAsBytes devuelve. Todos los campos están definidos como public static int.

Valor del campo	Descripción de los datos devueltos
REPRESENTATION_FIXED_STRING (0)	Datos de serie de longitud fija
REPRESENTATION_BIG_ENDIAN (1)	Formato binario con signo en el que se almacena el byte más significativo en la dirección más alta
REPRESENTATION_LITTLE_ENDIAN (2)	Formato binario con signo en el que se almacena el byte menos significativo en la dirección más alta
REPRESENTATION_VARIABLE_STRING (2)	Datos de serie que empiezan con un campo de dos bytes de longitud
REPRESENTATION_NUL_TERMINATED_STRING (3)	Datos de serie terminada en NULL
REPRESENTATION_FIXED_BYTES (4)	Serie de bytes de longitud fija
REPRESENTATION_VARIABLE_BYTES (5)	Serie de bytes que empieza con un campo de dos bytes de longitud

Valor del campo	Descripción de los datos devueltos
REPRESENTATION_NUL_TERMINATED_BYTES (7)	Datos de byte terminados en NULL
REPRESENTATION_FIXED_BINARY (15)	Serie binaria de longitud fija
REPRESENTATION_VARIABLE_BINARY (16)	Serie binaria que empieza con un campo de dos bytes de longitud
REPRESENTATION_PACKED_DECIMAL (48)	Serie binaria terminada en NULL
REPRESENTATION_NUMERIC_CHARACTER (50)	Formato de coma fija, basado en caracteres
REPRESENTATION_ZONED_DECIMAL (51)	Formato decimal con zona que IBM System i e IBM System z devuelven
REPRESENTATION_COBOL2_ZONED_DECIMAL (53)	Formato decimal con zona que los sistemas Windows o UNIX devuelven
REPRESENTATION_HEXADECIMAL_FLOATING_POINT (64)	Formato de coma flotante hexadecimal S/390
REPRESENTATION_DECIMAL_FLOATING_POINT (66)	Formato de coma flotante decimal
REPRESENTATION_IEEE_754_FLOATING_POINT_BYTE_REVERSED (71)	Formato de coma flotante IEEE en el que se almacena el byte menos significativo en la dirección más alta
REPRESENTATION_IEEE_754_FLOATING_POINT (72)	Formato de coma flotante IEEE en el que se almacena el byte más significativo en la dirección más alta

Métodos DB2ResultSet

Los métodos siguientes se definen únicamente para IBM Data Server Driver para JDBC y SQLJ.

getDB2RowChangeToken

Formato:

```
public long DB2ResultSet.getDB2RowChangeToken()
  throws java.sql.SQLException
```

Devuelve el símbolo de cambio de fila para la fila actual, si está disponible. Devuelve 0 si no se solicitaron columnas de bloqueo optimista o no están disponibles.

Este método se aplica únicamente a las conexiones con DB2 Database para Linux, UNIX y Windows.

getDB2RID

Formato:

```
public Object DB2ResultSet.getDB2RID()
  throws java.sql.SQLException
```

Devuelve el RID de la fila actual, si está disponible. Se puede obtener el RID si se solicitaron columnas de bloqueo optimista y están disponibles. Devuelve un valor nulo si no se solicitaron columnas de bloqueo optimista o no están disponibles.

Este método se aplica únicamente a las conexiones con DB2 Database para Linux, UNIX y Windows.

getDB2RIDType

Formato:

```
public int DB2ResultSet.getDB2RIDType()
  throws java.sql.SQLException
```

Devuelve el tipo de datos de la columna RID en un DB2ResultSet. El valor devuelto se correlaciona con una constante `java.sql.Types`. Si el DB2ResultSet no contiene una columna RID, se devuelve `java.sql.Types.NULL`.

Este método se aplica únicamente a las conexiones con DB2 Database para Linux, UNIX y Windows.

getDBRowDataAsBytes

Formato:

```
public Object [] getDBRowDataAsBytes()
  throws SQLException
```

Devuelve una matriz Object que representa los datos de la fila actual de un objeto ResultSet abierto.

Este método no es aplicable a las conexiones con IBM Informix.

No puede llamarse a getDBRowDataAsBytes si el objeto ResultSet en el que opera cumple cualquiera de las condiciones siguientes:

- Se ha actualizado, suprimido o insertado una fila de ResultSet que está recuperándose.
- El objeto ResultSet se ha creado para el bloqueo optimista.

La información devuelta incluye:

- Los datos en formato de matriz de bytes sin procesar
- El desplazamiento de los datos para cada columna

Imaginemos que obj es una instancia de la matriz Object devuelta. El formato de la matriz Object es:

obj[0] Una matriz de bytes que describe los datos de fila.

obj[1] Una matriz de enteros que contiene el desplazamiento en obj[0] de cada descripción de columna. Los desplazamientos pueden utilizarse para determinar la longitud de los datos que se devuelven para cada columna. Esa longitud representa la longitud de los datos sin procesar, y no la longitud definida de la columna.

Si un objeto ResultSet contiene una columna de cualquiera de los tipos siguientes, el valor de desplazamiento para ese valor de columna en obj[1] es -1. -1 indica que no se devuelve un valor para esa columna.
- BLOB
- CLOB
- DBCLOB
- XML

La matriz de bytes en obj[0] tiene el formato siguiente:

```
rnndd...dd...nndd...dd
```

Existe un conjunto *nndd...dd* para cada columna de la fila.

En la tabla siguiente se describe el contenido de los datos de fila:

Elemento	Descripción	
r	Un solo byte que tiene uno de los valores siguientes:	
	0	Los datos de fila no son válidos Una razón de la existencia de datos no válidos puede deberse a que todavía no se ha realizado la captación de la fila.
	1	Los datos de fila son válidos.

Elemento	Descripción
nn	Un indicador NULL de dos bytes para un valor de columna. Los valores posibles son: **-1** El valor de columna que sigue es NULL. **0** El valor de columna que sigue no es NULL.
dd...dd	Datos de byte sin procesar para un valor de columna.

getDBRowDescriptor

Formato:

```
public int [] getDBRowDescriptor()
  throws SQLException
```

Devuelve una matriz int que contiene información descriptiva acerca de cada columna de los datos de fila que getDBRowDataAsBytes devuelve.

Este método no es aplicable a las conexiones con IBM Informix.

Imaginemos que returnedInfo es una instancia de la matriz que getDBRowDescriptor devuelve. El formato de la matriz devuelta es:

returnedInfo[0]
El número de columnas de los datos de fila. Imaginemos que este valor es *n*.

returnedInfo[1] mediante returnedInfo[*4*n*]
n grupos que se repiten de valores de cuatro enteros. Cada grupo contiene información descriptiva para una única columna. Esa información es:

Número de descriptor de columna	Descripción
1	El tipo de datos de la columna, expresado como un valor SQLTYPE. Este valor es igual al valor SQLTYPE que se devuelve en una SQLDA.
2	El CCSID de la columna para un tipo de datos de caracteres. Para un tipo de datos DECIMAL, este valor es la escala de la columna.
3	La longitud definida de la columna, para todos los tipos de datos excepto DECIMAL. Para un tipo de datos DECIMAL, este valor es la precisión de la columna. Para los tipos de datos de caracteres de longitud variable, este valor podría ser mayor que el número de bytes devueltos.
4	Información adicional acerca de la columna. Los valores posibles se describen en el apartado “Campos de DB2ResultSet” en la página 568.

getDBTimestamp

Formatos:

```
public DBTimestamp getDBTimestamp(int
ÍndiceParámetro)
  throws SQLException
public DBTimestamp getDBTimestamp(String
NombreParámetro)
  throws SQLException
```

Devuelve el valor de la fila actual de una columna TIMESTAMP o TIMESTAMP WITH TIME ZONE que está en un objeto DB2ResultSet como

objeto DBTimestamp. Para una columna TIMESTAMP, el valor devuelto tiene el huso horario local. Si el valor de la columna DB2ResultSet es NULL, el valor que se devuelve es nulo.

Parámetros:

índiceParámetro
Número de la columna de DB2ResultSet cuyo valor se está recuperando.

nombreParámetro
Nombre de la columna de DB2ResultSet cuyo valor se está recuperando.

updateDBDefault

Formatos:

```
public void updateDBDefault(int índiceParámetro)
  throws SQLException
public void updateDBDefault(String nombreColumna)
  throws SQLException
```

Asigna el valor por omisión a la columna especificada en un objeto DB2ResultSet. Este método no actualiza la tabla subyacente.

Parámetros:

índiceParámetro
Número de la columna de DB2ResultSet cuyo valor se está actualizando.

nombreColumna
Nombre de la columna de DB2ResultSet cuyo valor se está actualizando.

Este método no se puede utilizar para conexiones con fuentes de datos IBM Informix.

Interfaz DB2ResultSetMetaData

La interfaz com.ibm.db2.jcc.DB2ResultSetMetaData proporciona métodos que proporcionan información sobre un objeto ResultSet.

Para poder utilizar un método com.ibm.db2.jcc.DB2ResultSetMetaData, se debe difundir a com.ibm.db2.jcc.DB2ResultSetMetaData un objeto java.sql.ResultSetMetaData que se devuelva de una llamada a java.sql.ResultSet.getMetaData.

Métodos de DB2ResultSetMetaData:

Los métodos siguientes se definen únicamente para IBM Data Server Driver para JDBC y SQLJ.

getDB2OptimisticLockingColumns

Formato:

```
public int getDB2OptimisticLockingColumns()
  throws java.sql.SQLException
```

Devuelve un valor que indica si están disponibles columnas de bloqueo optimista. Los valores posibles son:

0 No existen columnas de bloqueo optimista disponibles.

1 Existen columnas de bloqueo optimista disponibles, pero el símbolo de cambio puede no tener la granularidad necesaria para evitar negativos falsos.

2 Existen columnas de bloqueo optimista disponibles, y el símbolo de cambio tiene la granularidad necesaria para evitar negativos falsos.

isDB2ColumnNameDerived

Formato:

```
public boolean isDB2ColumnNameDerived (int columna)
  throws java.sql.SQLException
```

Devuelve el valor `true` si el nombre de la columna ResultSet está en la lista de SQL SELECT que ha generado el conjunto de resultados ResultSet.

Por ejemplo, supongamos que se genera un conjunto de resultados (ResultSet) a partir de la sentencia de SQL SELECT EMPNAME, SUM(SALARY) FROM EMP. El nombre de columna EMPNAME se deriva de la lista de SQL SELECT, pero el nombre de la columna del conjunto de resultados ResultSet que corresponde a SUM(SALARY) no se deriva de la lista de SELECT.

Descripciones de parámetros:

columna
La posición ordinal de una columna en ResultSet.

getDBTemporalColumnType

Formato:

```
public int getDBTemporalColumnType (int columna)
  throws java.sql.SQLException
```

Devuelve:

-1 Si *columna* no es una columna ROW BEGIN, ROW END o TRANSACTION START ID.

1 Si *columna* es una columna ROW BEGIN.

2 Si *columna* es una columna ROW END.

3 Si *columna* es una columna TRANSACTION START ID.

Descripciones de parámetros:

columna
La posición ordinal de una columna en ResultSet.

Interfaz DB2RowID

La interfaz com.ibm.db2.jcc.DB2RowID se utiliza para declarar objetos Java para su utilización con el tipo de datos ROWID de SQL.

La interfaz com.ibm.db2.jcc.DB2RowID no es aplicable a la conexión con IBM Informix.

Métodos DB2RowID

El método siguiente sólo se define para IBM Data Server Driver para JDBC y SQLJ.

getBytes

Formato:

```
public byte[] getBytes()
```

Convierte un objeto com.ibm.jcc.DB2RowID a bytes.

Clase DB2SimpleDataSource

La clase com.ibm.db2.jcc.DB2SimpleDataSource amplía la clase DB2BaseDataSource.

Un objeto DB2BaseDataSource no da soporte a la agrupación de conexiones ni a las transacciones distribuidas. Contiene todas las propiedades y los métodos contenidos por la clase DB2BaseDataSource. Además, DB2SimpleDataSource contiene las siguientes propiedades específicas de IBM Data Server Driver para JDBC y SQLJ.

DB2SimpleDataSource implementa la interfaz java.sql.Wrapper.

Métodos DB2SimpleDataSource

El método siguiente sólo se define para IBM Data Server Driver para JDBC y SQLJ.

setPassword

Formato:

```
public synchronized void setPassword(String contraseña)
```

Establece la contraseña para el objeto DB2SimpleDataSource. No existe un método correspondiente para getPassword. Por tanto, la contraseña no se puede cifrar, pues no hay forma de recuperar la contraseña para poder descifrarla.

Clase DB2Sqlca

La clase com.ibm.db2.jcc.DB2Sqlca es una encapsulación de la SQLCA.

Métodos DB2Sqlca

Los métodos siguientes se definen únicamente para IBM Data Server Driver para JDBC y SQLJ.

getMessage

Formato:

```
public abstract String getMessage()
```

Devuelve el texto de mensajes de error.

getSqlCode

Formato:

```
public abstract int getSqlCode()
```

Devuelve el valor de un código de error de SQL.

getSqlErrd

Formato:

```
public abstract int[] getSqlErrd()
```

Devuelve una matriz, cuyos elementos contienen un SQLERRD de SQLCA.

getSqlErrmc

Formato:

```
public abstract String getSqlErrmc()
```

Devuelve una serie de caracteres que contiene los valores SQLERRMC de SQLCA, delimitados por espacios.

getSqlErrmcTokens

Formato:

```
public abstract String[] getSqlErrmcTokens()
```

Devuelve una matriz, cuyos elementos contienen un símbolo SQLERRMC de SQLCA.

getSqlErrp

Formato:

```
public abstract String getSqlErrp()
```

Devuelve el valor SQLERRP de SQLCA.

getSqlState

Formato:

```
public abstract String getSqlState()
```

Devuelve el valor SQLSTATE de SQLCA.

getSqlWarn

Formato:

```
public abstract char[] getSqlWarn()
```

Devuelve una matriz, cuyos elementos contienen un valor SQLWARN de SQLCA.

Interfaz DB2Statement

La interfaz com.ibm.db2.jcc.DB2Statement amplía la interfaz java.sql.Statement.

DB2Statement implementa la interfaz java.sql.Wrapper.

Campos de DB2Statement

Los campos siguientes se definen solamente para IBM Data Server Driver para JDBC y SQLJ.

```
public static final int RETURN_OPTLOCK_COLUMN_NONE = 0
public static final int RETURN_OPTLOCK_COLUMN_ALWAYS = 1
public static final int RETURN_OPTLOCK_COLUMN_NO_FALSE_NEGATIVES = 2
```

Estos valores son argumentos del método DB2Statement.executeDB2OptimisticLockingQuery.

Métodos DB2Statement

Los métodos siguientes se definen únicamente para IBM Data Server Driver para JDBC y SQLJ.

executeDB2OptimisticLockingQuery

Formato:

```
public java.sql.ResultSet DB2Statement.executeDB2OptimisticLockingQuery(
  String sql,
  int returnOptLockingColumn)
  throws java.sql.SQLException
```

Ejecuta una sentencia de una consulta de SQL y devuelve un ResultSet que contiene información de bloqueo optimista, si se solicita.

Descripciones de parámetros:

sql
Sentencia SELECT de SQL que devuelve un ResultSet individual.

returnOptimisticLockingColumns
Especifica si se devuelven columnas de bloqueo optimista. Los valores posibles son:

Tabla 113. Valores para el parámetro returnOptimisticLockingColumns

Valor	Descripción
DB2Statement.RETURN_OPTLOCK_COLUMN_NONE (0)	No devolver columnas de bloqueo optimista.
DB2Statement.RETURN_OPTLOCK_COLUMN_ALWAYS (1)	Añade columnas de cambio de fila al conjunto de resultados incluso si no representan de forma exclusiva una única fila. Este valor es equivalente al atributo de preparación de base de datos WITH ROW CHANGE COLUMNS POSSIBLY DISTINCT.
DB2Statement.RETURN_OPTLOCK_COLUMN_NO_FALSE_NEGATIVES (2)	Añade columnas de cambio de fila al conjunto de resultados únicamente si representan una única fila. Este valor es equivalente al atributo de preparación de base de datos WITH ROW CHANGE COLUMNS ALWAYS DISTINCT.

getAffectedRowCount
Formato:

```
public int getAffectedRowCount()
  throws java.sql.SQLException
```

Devuelve el número de filas que estén afectadas por la correcta ejecución de una sentencia de SQL. Si la sentencia de SQL es INSERT, UPDATE o DELETE, getAffectedRowCount devuelve el mismo valor devuelto por java.sql.Statement.getUpdateCount.

El valor devuelto por getAffectedRowCount es la misma información devuelta por el servidor de datos en la SQLCA tras la correcta ejecución de una sentencia de SQL.

getDB2ClientProgramId
Formato:

```
public String getDB2ClientProgramId()
  throws java.sql.SQLException
```

Devuelve el identificador del programa cliente definido por el usuario correspondiente a la conexión, que está almacenado en la fuente de datos.

getDB2ClientProgramId no se aplica a los servidores de datos de DB2 Database para Linux, UNIX y Windows.

setDB2ClientProgramId
Formato:

```
public abstract void setDB2ClientProgramId(String program-ID)
  throws java.sql.SQLException
```

Establece un identificador de programa definido por el usuario correspondiente a la conexión en un servidor de datos. Dicho identificador de programa es una serie de 80 bytes que se utiliza para identificar al llamador.

setDB2ClientProgramId no se aplica a servidores de datos DB2 Database para Linux, UNIX y Windows.

El servidor DB2 para z/OS coloca la serie en los registros de rastreo IFCID 316 junto con los demás datos estadísticos, de modo que se pueda identificar el programa que está asociado con una sentencia de SQL determinada.

getIDSBigSerial

Formato:

```
public int getIDSBigSerial()
  throws java.sql.SQLException
```

Recupera una clave que se ha generado automáticamente en la columna BIGSERIAL después de que una sentencia INSERT ejecutada anteriormente haya insertado la clave generada automáticamente.

Se deben cumplir las condiciones siguientes para que getIDSBigSerial se ejecute satisfactoriamente:

- La sentencia INSERT es la última sentencia de SQL que se ha ejecutado antes de invocar este método.
- La tabla en la que se inserta la fila contiene una columna BIGSERIAL.
- El formato del método Connection.prepareStatement o Statement.executeUpdate de JDBC por el que se prepara o ejecuta la sentencia INSERT no tiene parámetros que solicitan claves de generación automática.

Este método se aplica únicamente a las conexiones con bases de datos IBM Informix.

getIDSSerial

Formato:

```
public int getIDSSerial()
  throws java.sql.SQLException
```

Recupera una clave que se ha generado automáticamente en la columna SERIAL después de que una sentencia INSERT ejecutada anteriormente haya insertado la clave generada automáticamente.

Se deben cumplir las condiciones siguientes para que getIDSSerial se ejecute satisfactoriamente:

- La sentencia INSERT es la última sentencia de SQL que se ha ejecutado antes de invocar este método.
- La tabla en la que se inserta la fila contiene una columna SERIAL.
- El formato del método Connection.prepareStatement o Statement.executeUpdate de JDBC por el que se prepara o ejecuta la sentencia INSERT no tiene parámetros que solicitan claves de generación automática.

Este método se aplica únicamente a las conexiones con bases de datos IBM Informix.

getIDSSerial8

Formato:

```
public long getIDSSerial8()
  throws java.sql.SQLException
```

Recupera una clave que se ha generado automáticamente en la columna SERIAL8 después de que una sentencia INSERT ejecutada anteriormente haya insertado la clave generada automáticamente.

Se deben cumplir las condiciones siguientes para que getIDSSerial8 se ejecute satisfactoriamente:

- La sentencia INSERT es la última sentencia de SQL que se ha ejecutado antes de invocar este método.
- La tabla en la que se inserta la fila contiene una columna SERIAL8.
- El formato del método Connection.prepareStatement o Statement.executeUpdate de JDBC por el que se prepara o ejecuta la sentencia INSERT no tiene parámetros que solicitan claves de generación automática.

Este método solamente es aplicable a conexiones con fuentes de datos IBM Informix.

getIDSSQLStatementOffSet

Formato:

```
public int getIDSSQLStatementOffSet()
  throws java.sql.SQLException
```

Después de que una sentencia de SQL se ejecute en una fuente de datos IBM Informix, si la sentencia contiene un error de sintaxis, getIDSSQLStatementOffSet devuelve el desplazamiento dentro del texto de la sentencia del error de sintaxis.

getIDSSQLStatementOffSet devuelve:

- 0, si la sentencia no contiene un error de sintaxis.
- -1, si la fuente de datos no es IBM Informix.

Este método solamente es aplicable a conexiones con fuentes de datos IBM Informix.

Interfaz DB2SystemMonitor

La interfaz com.ibm.db2.jcc.DB2SystemMonitor se utiliza para recoger datos de supervisión del sistema correspondientes a una conexión. Cada conexión puede tener una sola instancia de DB2SystemMonitor.

Campos de DB2SystemMonitor

Los campos siguientes se definen solamente para IBM Data Server Driver para JDBC y SQLJ.

public final static int RESET_TIMES
public final static int ACCUMULATE_TIMES

Estos valores son argumentos del método DB2SystemMonitor.start. RESET_TIMES inicializa a cero los contadores de tiempo antes del comienzo de la supervisión. ACCUMULATE_TIMES no pone a cero los contadores de tiempo.

Métodos DB2SystemMonitor

Los métodos siguientes se definen únicamente para IBM Data Server Driver para JDBC y SQLJ.

enable

Formato:

```
public void enable(boolean on)
  throws java.sql.SQLException
```

Habilita el supervisor del sistema que está asociado a una conexión. Este método no se puede invocar durante la supervisión. Todos los tiempos se inicializan cuando se invoca enable.

getApplicationTimeMillis

Formato:

```
public long getApplicationTimeMillis()
  throws java.sql.SQLException
```

Devuelve la suma de los tiempos transcurridos correspondientes a la aplicación, el controlador JDBC, la E/S de red y el servidor de bases de datos. El tiempo está expresado en milisegundos.

Un intervalo de tiempo transcurrido supervisado es la diferencia, en milisegundos, entre estos puntos en el proceso del controlador JDBC:

Inicio del intervalo
Es el momento en el que se invoca start.

Fin del intervalo
Es el momento en el que se invoca stop.

getApplicationTimeMillis devuelve un 0 si la supervisión del sistema está inhabilitada. Si se invoca este método sin invocar primero stop, se produce una excepción SQLException.

getCoreDriverTimeMicros

Formato:

```
public long getCoreDriverTimeMicros()
  throws java.sql.SQLException
```

Devuelve la suma de los tiempos transcurridos de API supervisada que se recogieron mientras la supervisión del sistema estaba habilitada. El tiempo está expresado en microsegundos.

Una API supervisada es un método de controlador JDBC para la cual se obtiene el tiempo de proceso. En general, los tiempos transcurridos sólo se supervisan para las API que puedan producir interacción con la E/S de red o servidor de bases de datos. Por ejemplo, los métodos PreparedStatement.setXXX y ResultSet.getXXX no se supervisan.

El tiempo transcurrido de las API supervisadas incluye el tiempo total que se invierte en el controlador para una llamada de método. Este tiempo incluye cualquier tiempo de E/S de red y tiempo transcurrido de servidor de bases de datos.

Un intervalo de tiempo transcurrido de API supervisada es la diferencia, en microsegundos, entre estos puntos en el proceso del controlador JDBC:

Inicio del intervalo
Es el momento en el que la aplicación llama a la API supervisada.

Fin del intervalo
Inmediatamente antes de este momento la API supervisada devuelve el control a la aplicación.

getCoreDriverTimeMicros devuelve un 0 si la supervisión del sistema está inhabilitada. Si se invoca este método sin primero llamar al método stop, o si se invoca este método cuando la JVM subyacente no da soporte a la notificación de tiempos en microsegundos, se produce una excepción SQLException.

getNetworkIOTimeMicros

Formato:

```
public long getNetworkIOTimeMicros()
  throws java.sql.SQLException
```

Devuelve la suma de los tiempos transcurridos de E/S de red que se recogieron mientras la supervisión del sistema estaba habilitada. El tiempo está expresado en microsegundos.

El tiempo transcurrido de E/S de red incluye el tiempo que se invierte en escribir y leer datos de DRDA procedentes de corrientes de E/S de la red. Un intervalo de tiempo transcurrido de E/S de red es el intervalo de tiempo que se invierte en realizar las operaciones siguientes en el controlador JDBC:

- Emitir un mandato TCP/IP para enviar un mensaje de DRDA al servidor de bases de datos. Este intervalo de tiempo es la diferencia, en microsegundos, entre los momentos inmediatamente anterior y posterior a la realización de una escritura y vaciado en la corriente de E/S de la red.
- Emitir un mandato TCP/IP para recibir mensajes de respuesta de DRDA procedentes del servidor de bases de datos. Este intervalo de tiempo es la diferencia, en microsegundos, entre los momentos inmediatamente anterior y posterior a la realización de una lectura en la corriente de E/S de la red.

Los intervalos de tiempo de E/S de red se recogen para todas las operaciones de envío y recepción, incluido el envío de mensajes para operaciones de confirmación y retrotracción.

El tiempo invertido en la espera a causa de operaciones de E/S de red puede verse afectado por retrasos en la CPU que despacha peticiones de SQL de baja prioridad en el servidor de bases de datos.

getNetworkIOTimeMicros devuelve un 0 si la supervisión del sistema está inhabilitada. Si se invoca este método sin primero llamar al método stop, o si se invoca este método cuando la JVM subyacente no da soporte a la notificación de tiempos en microsegundos, se produce una excepción SQLException.

getServerTimeMicros

Formato:

```
public long getServerTimeMicros()
  throws java.sql.SQLException
```

Devuelve la suma de todos los tiempos transcurridos notificados del servidor de bases de datos que se recopilaron mientras la supervisión del sistema estaba habilitada. El tiempo está expresado en microsegundos.

El servidor de bases de datos notifica tiempos transcurridos cuando se dan estas condiciones:

- El servidor de bases de datos da soporte a la devolución de datos sobre tiempos transcurridos al cliente.

  DB2 Database para Linux, UNIX y Windows versión 9.5 y posterior, y DB2 para z/OS son compatibles con esta función.
- El servidor de bases de datos efectúa operaciones que se pueden supervisar. Por ejemplo, el tiempo transcurrido del servidor de bases de datos no se devuelve para operaciones de confirmación ni retrotracción.

Para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 con DB2 Database para Linux, UNIX y Windows, y IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4: El tiempo transcurrido del servidor de bases de

datos se define como el tiempo transcurrido para analizar la corriente de datos de la petición, procesar el mandato y generar la corriente de datos de respuesta en el servidor de bases de datos. El tiempo de red necesario para recibir o enviar la corriente de datos no se incluye. Un intervalo de tiempo transcurrido del servidor de bases de datos es la diferencia, en microsegundos, entre estos puntos en el proceso del servidor:

Inicio del intervalo
Cuando el sistema operativo hace que el servidor de bases de datos procese un mensaje TCP/IP que se recibe del controlador JDBC.

Fin del intervalo
Cuando el servidor de bases de datos está preparado para emitir el mandato de TCP/IP para devolver el mensaje de respuesta al cliente.

Para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 con DB2 para z/OS: El intervalo de tiempo transcurrido del servidor de bases de datos es la diferencia, en microsegundos, entre estos puntos en el proceso nativo del controlador JDBC:

Inicio del intervalo
Valor STCK (Store Clock) de z/OS cuando un método nativo del controlador JDBC invoca el recurso de conexión RRS para procesar una petición de SQL.

Fin del intervalo
Valor de STCK (Store Clock) de z/OS cuando el control vuelve al método nativo del controlador JDBC después de una llamada al recurso de conexión RRS para procesar una petición de SQL.

getServerTimeMicros devuelve un 0 si la supervisión del sistema está inhabilitada. Si se invoca este método sin invocar primero stop, se produce una excepción SQLException.

start

Formato:

```
public void start (int lapMode)
  throws java.sql.SQLException
```

Si el supervisor del sistema está habilitado, start inicia la recogida de datos de supervisión del sistema para una conexión. Los valores válidos de *lapMode* son RESET_TIMES o ACCUMULATE_TIMES.

La invocación de este método cuando la supervisión del sistema está inhabilitada no produce ningún efecto. La invocación de este método más de una vez sin que medie una llamada a stop produce una excepción SQLException.

stop

Formato:

```
public void stop()
  throws java.sql.SQLException
```

Si el supervisor del sistema está habilitado, stop finaliza la recogida de datos de supervisión del sistema para una conexión. Una vez detenida la supervisión, se pueden obtener los tiempos supervisados utilizando los métodos getXXX de DB2SystemMonitor.

La invocación de este método cuando la supervisión del sistema está inhabilitada no produce ningún efecto. La invocación de este método sin

primero invocar start, o la invocación repetida de este método sin que medie una llamada a start produce una excepción SQLException.

Clase DB2TraceManager

La clase com.ibm.db2.jcc.DB2TraceManager controla el grabador de anotaciones globales.

El grabador de anotaciones globales se aplica a todas las conexiones en todo el controlador. El grabador de anotaciones globales altera temporalmente todos los demás grabadores de anotaciones globales de JDBC. Además de iniciar el grabador de anotaciones globales, la clase DB2TraceManager proporciona la capacidad de suspender y reanudar el rastreo de cualquier tipo de grabador de anotaciones. Es decir, los métodos de suspensión y reanudación de la clase DB2TraceManager son aplicables a todos los programas de anotaciones cronológicas actuales y futuros de DriverManager, programas de anotaciones cronológicas de DataSource o programas de anotaciones cronológicas a nivel de conexión específicos de IBM Data Server Driver para JDBC y SQLJ.

Métodos DB2TraceManager

getTraceManager

Formato:

```
static public DB2TraceManager getTraceManager()
  throws java.sql.SQLException
```

Obtiene una instancia del grabador de anotaciones cronológicas globales.

setLogWriter

Formatos:

```
public abstract void setLogWriter(String traceDirectory,
  String baseTraceFileName, int traceLevel)
  throws java.sql.SQLException
public abstract void setLogWriter(String traceFile,
  boolean fileAppend, int traceLevel)
  throws java.sql.SQLException
public abstract void setLogWriter(java.io.PrintWriter logWriter,
  int traceLevel)
  throws java.sql.SQLException
```

Habilita un rastreo global. Después de llamar a setLogWriter, se descartan todas las llamadas para los rastreos de DataSource o Connection hasta que se llame a DB2TraceManager.unsetLogWriter.

Cuando se llama a setLogWriter, se redireccionan todos los rastreos de Connection o DataSource a un archivo de rastreo o PrintWriter, en función del formato de setLogWriter que se utilice. Si se suspende el rastreo global cuando se llama a setLogWriter, los valores especificados surten efecto cuando se reanuda el rastreo.

Descripciones de parámetros:

traceDirectory

Especifica un directorio en el que se graba la información de rastreo global. Este valor prevalece sobre los valores de las propiedades traceDirectory y logWriter para una conexión DataSource o DriverManager.

Cuando se utiliza el formato de setLogWriter con el parámetro traceDirectory, el controlador de JDBC establece la propiedad traceFileAppend en `false` cuando se llama a setLogWriter, lo que significa que se sobregraban los archivos de anotaciones existentes. Cada conexión

de controlador de JDBC se rastrea en un archivo diferente en el directorio especificado. El convenio de denominación para los archivos de dicho directorio depende de si se especifica un valor no nulo para baseTraceFileName:

- Si se especifica un valor nulo para baseTraceFileName, se rastrea una conexión para un archivo denominado traceFile_global_*n*.

  n es la conexión de controlador de JDBC número *n*.
- Si se especifica un valor no nulo para baseTraceFileName, se rastrea una conexión para un archivo denominado *baseTraceFileName*_global_*n*.

  baseTraceFileName es el valor del parámetro baseTraceFileName.

  n es la conexión de controlador de JDBC número *n*.

baseTraceFileName
Especifica la raíz de los nombres de los archivos en los que se graba la información de rastreo global. La combinación de baseTraceFileName y traceDirectory determina el nombre de vía de acceso completa para los archivos de anotaciones de rastreo globales.

traceFileName
Especifica el archivo en el que se graba la información de rastreo global. Este valor prevalece sobre los valores de las propiedades traceFile y logWriter para una conexión DataSource o DriverManager.

Cuando se utiliza el formato de setLogWriter con el parámetro traceFileName, se escribe un solo archivo de anotaciones cronológicas.

traceFileName puede incluir una vía de acceso de directorio.

logWriter
Especifica una corriente de salida de caracteres en el que se graban todos los registros de anotaciones globales.

Este valor prevalece sobre la propiedad logWriter en una conexión DataSource o DriverManager.

traceLevel
Especifica qué se debe rastrear.

Puede especificar uno o más de los valores de rastreo siguientes con el parámetro traceLevel:

- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_NONE (X'00')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_CONNECTION_CALLS (X'01')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_STATEMENT_CALLS (X'02')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_RESULT_SET_CALLS (X'04')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DRIVER_CONFIGURATION (X'10')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_CONNECTS (X'20')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DRDA_FLOWS (X'40')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_RESULT_SET_META_DATA (X'80')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_PARAMETER_META_DATA (X'100')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DIAGNOSTICS (X'200')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_SQLJ (X'400')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_XA_CALLS (sólo IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 para DB2 Database para Linux, UNIX y Windows) (X'800')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_META_CALLS (X'2000')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DATASOURCE_CALLS (X'4000')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_LARGE_OBJECT_CALLS (X'8000')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_SYSTEM_MONITOR (X'20000')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_TRACEPOINTS () (X'40000')

- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL` (X'FFFFFFFF')

Para especificar más de un valor de rastreo, utilice una de estas técnicas:

- Utilice operadores OR (`|`) de bits con dos o más valores de rastreo. Por ejemplo, para rastrear flujos de DRDA y llamadas de conexión, especifique este valor para traceLevel:

  ```
  TRACE_DRDA_FLOWS|TRACE_CONNECTION_CALLS
  ```

- Utilice un operador de complemento a nivel de bit (tilde (~)) con un valor de rastreo para especificar todos los rastreos excepto uno determinado. Por ejemplo, para rastrear todo excepto los flujos de DRDA, especifique este valor para traceLevel:

  ```
  ~TRACE_DRDA_FLOWS
  ```

fileAppend
Especifica si se deben añadir o sobrescribir datos en el archivo especificado por medio del parámetro traceFile. `true` significa que no se sobrescribirá el archivo existente.

unsetLogWriter
Formato:

```
public abstract void unsetLogWriter()
  throws java.sql.SQLException
```

Inhabilita la alteración temporal del grabador de anotaciones globales para conexiones futuras.

suspendTrace
Formato:

```
public void suspendTrace()
  throws java.sql.SQLException
```

Suspende todos los rastreos globales a nivel de Connection o a nivel de DataSource para las conexiones actuales y futuras. suspendTrace puede llamarse cuando se habilita o inhabilita el grabador de anotaciones globales.

resumeTrace
Formato:

```
public void resumeTrace()
  throws java.sql.SQLException
```

Reanuda todos los rastreos globales a nivel de Connection o a nivel de DataSource para las conexiones actuales y futuras. resumeTrace puede llamarse cuando se habilita o inhabilita el grabador de anotaciones globales. Si se inhabilita el grabador de anotaciones globales, resumeTrace reanuda los rastreos a nivel de Connection o a nivel de DataSource. Si se habilita el grabador de anotaciones globales, resumeTrace reanuda el rastreo global.

getLogWriter
Formato:

```
public abstract java.io.PrintWriter getLogWriter()
  throws java.sql.SQLException
```

Devuelve el PrintWriter para el grabador de anotaciones globales, si se establece éste. En caso contrario, getLogWriter devuelve un nulo.

getTraceFile
Formato:

```
public abstract String getTraceFile()
  throws java.sql.SQLException
```

Devuelve el nombre del archivo de destino para el grabador de anotaciones globales, si se establece éste. En otro caso, getTraceFile devuelve un valor nulo.

getTraceDirectory

Formato:

```
public abstract String getTraceDirectory()
  throws java.sql.SQLException
```

Devuelve el nombre del directorio de destino para los archivos del grabador de anotaciones globales, si se establece éste. En caso contrario, getTraceDirectory devuelve un nulo.

getTraceLevel

Formato:

```
public abstract int getTraceLevel()
  throws java.sql.SQLException
```

Devuelve el nivel de rastreo para el rastreo global, si se establece éste. De lo contrario, getTraceLevel devuelve -1 (`TRACE_ALL`).

getTraceFileAppend

Formato:

```
public abstract boolean getTraceFileAppend()
  throws java.sql.SQLException
```

Devuelve `true` si los registros de rastreo globales se agregan al archivo de rastreo. De lo contrario, getTraceFileAppend devuelve `false`.

Interfaz DB2TraceManagerMXBean

La interfaz com.ibm.db2.jcc.mx.DB2TraceManagerMXBean es el medio utilizado por una aplicación para hacer que DB2TraceManager esté disponible como MXBean para el controlador de rastreo remoto.

Métodos DB2TraceManagerMXBean

setTraceFile

Formato:

```
public void setTraceFile(String traceFile,
  boolean fileAppend, int traceLevel)
  throws java.sql.SQLException
```

Especifica el nombre del archivo donde el gestor de rastreo remoto escribe información de rastreo, y el tipo de información que se debe rastrear.

Descripciones de parámetros:

traceFileName
Especifica el archivo en el que se graba la información de rastreo global. Este valor prevalece sobre los valores de las propiedades traceFile y logWriter para una conexión DataSource o DriverManager.

Cuando se utiliza el formato de setLogWriter con el parámetro traceFileName, se escribe un solo archivo de anotaciones cronológicas.

traceFileName puede incluir una vía de acceso de directorio.

fileAppend
Especifica si se deben añadir o sobrescribir datos en el archivo especificado por medio del parámetro traceFile. `true` significa que no se sobrescribirá el archivo existente.

traceLevel
Especifica qué se debe rastrear.

Puede especificar uno o más de los valores de rastreo siguientes con el parámetro traceLevel:
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_NONE (X'00')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_CONNECTION_CALLS (X'01')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_STATEMENT_CALLS (X'02')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_RESULT_SET_CALLS (X'04')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DRIVER_CONFIGURATION (X'10')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_CONNECTS (X'20')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DRDA_FLOWS (X'40')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_RESULT_SET_META_DATA (X'80')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_PARAMETER_META_DATA (X'100')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DIAGNOSTICS (X'200')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_SQLJ (X'400')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_XA_CALLS (sólo IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 para DB2 Database para Linux, UNIX y Windows) (X'800')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_META_CALLS (X'2000')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DATASOURCE_CALLS (X'4000')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_LARGE_OBJECT_CALLS (X'8000')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_SYSTEM_MONITOR (X'20000')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_TRACEPOINTS () (X'40000')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL (X'FFFFFFFF')

Para especificar más de un valor de rastreo, utilice una de estas técnicas:
- Utilice operadores OR (|) de bits con dos o más valores de rastreo. Por ejemplo, para rastrear flujos de DRDA y llamadas de conexión, especifique este valor para traceLevel:

  ```
  TRACE_DRDA_FLOWS|TRACE_CONNECTION_CALLS
  ```
- Utilice un operador de complemento a nivel de bit (tilde (~)) con un valor de rastreo para especificar todos los rastreos excepto uno determinado. Por ejemplo, para rastrear todo excepto los flujos de DRDA, especifique este valor para traceLevel:

  ```
  ~TRACE_DRDA_FLOWS
  ```

getTraceFile
Formato:

```
public void getTraceFile()
  throws java.sql.SQLException
```

Devuelve el nombre del archivo de destino para el controlador de rastreo remoto, si está establecido. En otro caso, getTraceFile devuelve un valor nulo.

setTraceDirectory
Formato:

```
public void setTraceDirectory(String traceDirectory,
  String baseTraceFileName,
  int traceLevel) throws java.sql.SQLException
```

Especifica el nombre del directorio donde el controlador de rastreo remoto escribe información de rastreo, y el tipo de información que se debe rastrear.

Descripciones de parámetros:

traceDirectory
Especifica un directorio en el que se graba la información de rastreo. Este

valor prevalece sobre los valores de las propiedades traceDirectory y logWriter para una conexión DataSource o DriverManager.

Cada conexión de controlador de JDBC se rastrea en un archivo diferente en el directorio especificado. El convenio de denominación para los archivos de dicho directorio depende de si se especifica un valor no nulo para baseTraceFileName:

- Si se especifica un valor nulo para baseTraceFileName, se rastrea una conexión para un archivo denominado traceFile_global_*n*.

  n es la conexión de controlador de JDBC número *n*.
- Si se especifica un valor no nulo para baseTraceFileName, se rastrea una conexión para un archivo denominado *baseTraceFileName*_global_*n*.

  baseTraceFileName es el valor del parámetro baseTraceFileName.

  n es la conexión de controlador de JDBC número *n*.

baseTraceFileName
Especifica la raíz de los nombres de los archivos en los que se graba la información de rastreo global. La combinación de baseTraceFileName y traceDirectory determina el nombre de vía de acceso completa para los archivos de anotaciones de rastreo globales.

traceLevel
Especifica qué se debe rastrear.

Puede especificar uno o más de los valores de rastreo siguientes con el parámetro traceLevel:

- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_NONE` (X'00')
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_CONNECTION_CALLS` (X'01')
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_STATEMENT_CALLS` (X'02')
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_RESULT_SET_CALLS` (X'04')
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DRIVER_CONFIGURATION` (X'10')
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_CONNECTS` (X'20')
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DRDA_FLOWS` (X'40')
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_RESULT_SET_META_DATA` (X'80')
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_PARAMETER_META_DATA` (X'100')
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DIAGNOSTICS` (X'200')
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_SQLJ` (X'400')
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_XA_CALLS (sólo IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 para DB2 Database para Linux, UNIX y Windows)` (X'800')
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_META_CALLS` (X'2000')
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DATASOURCE_CALLS` (X'4000')
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_LARGE_OBJECT_CALLS` (X'8000')
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_SYSTEM_MONITOR` (X'20000')
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_TRACEPOINTS` () (X'40000')
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL` (X'FFFFFFFF')

Para especificar más de un valor de rastreo, utilice una de estas técnicas:

- Utilice operadores OR (`|`) de bits con dos o más valores de rastreo. Por ejemplo, para rastrear flujos de DRDA y llamadas de conexión, especifique este valor para traceLevel:

  ```
  TRACE_DRDA_FLOWS|TRACE_CONNECTION_CALLS
  ```
- Utilice un operador de complemento a nivel de bit (tilde (~)) con un valor de rastreo para especificar todos los rastreos excepto uno determinado. Por ejemplo, para rastrear todo excepto los flujos de DRDA, especifique este valor para traceLevel:

~TRACE_DRDA_FLOWS

getTraceFileAppend

Formato:

```
public abstract boolean getTraceFileAppend()
  throws java.sql.SQLException
```

Devuelve `true` si los registros de rastreo producidos por el controlador de rastreo se añaden al archivo de rastreo. De lo contrario, getTraceFileAppend devuelve `false`.

getTraceDirectory

Formato:

```
public void getTraceDirectory()
  throws java.sql.SQLException
```

Devuelve el nombre del directorio de destino para los registros de rastreo producidos por el controlador de rastreo, si se directorio está establecido. En caso contrario, getTraceDirectory devuelve un nulo.

getTraceLevel

Formato:

```
public void getTraceLevel()
  throws java.sql.SQLException
```

Devuelve el nivel de rastreo para los registros de rastreo producidos por el controlador de rastreo, si ese valor está establecido. En otro caso, getTraceLevel devuelve -1 (`TRACE_ALL`).

unsetLogWriter

Formato:

```
public abstract void unsetLogWriter()
  throws java.sql.SQLException
```

Inhabilita la alteración temporal del grabador de anotaciones globales para conexiones futuras.

suspendTrace

Formato:

```
public void suspendTrace()
  throws java.sql.SQLException
```

Suspende todos los rastreos globales a nivel de Connection o a nivel de DataSource para las conexiones actuales y futuras. suspendTrace puede llamarse cuando se habilita o inhabilita el grabador de anotaciones globales.

resumeTrace

Formato:

```
public void resumeTrace()
  throws java.sql.SQLException
```

Reanuda todos los rastreos globales a nivel de Connection o a nivel de DataSource para las conexiones actuales y futuras. resumeTrace puede llamarse cuando se habilita o inhabilita el grabador de anotaciones globales. Si se inhabilita el grabador de anotaciones globales, resumeTrace reanuda los rastreos a nivel de Connection o a nivel de DataSource. Si se habilita el grabador de anotaciones globales, resumeTrace reanuda el rastreo global.

Interfaz DB2Struct

La interfaz com.ibm.db2.jcc.DB2Struct proporciona métodos específicos de IBM Data Server Driver para JDBC y SQLJ para trabajar con objetos Struct.

Métodos de DB2Struct

getMetaData

Formato:

```
java.sql.ResultSetMetaData getMetaData()
  throws SQLException
```

Devuelve metadatos para un objeto DB2Struct.

Clase DB2Types

La clase com.ibm.db2.jcc.DB2Types proporciona campos que definen tipos de datos únicamente de IBM Data Server Driver para JDBC y SQLJ.

Campos DB2Types

Las siguientes constantes definen tipos de códigos solamente para IBM Data Server Driver para JDBC y SQLJ.

- public final static int BLOB_FILE = -100002
- public final static int CLOB_FILE = -100003
- public final static int CURSOR = -100008
- public final static int DECFLOAT = -100001
- public final static int XML_AS_BLOB_FILE = -100004
- public final static int XML_AS_CLOB_FILE = -100005
- public final static int TIMESTAMPTZ =-100010

Clase DB2XADataSource

DB2XADataSource es una fábrica para los objetos XADataSource. Un objeto que implementa esta interfaz se registra con un servicio de asignación de nombres que se basa en Java Naming and Directory Interface (JNDI).

La clase com.ibm.db2.jcc.DB2XADataSource amplía la clase com.ibm.db2.jcc.DB2BaseDataSource e implementa las interfaces javax.sql.XADataSource, java.io.Serializable y javax.naming.Referenceable.

Métodos DB2XADataSource

getDB2TrustedXAConnection

Formatos:

```
public Object[] getDB2TrustedXAConnection(String user,
  String password,
  java.util.Properties properties)
  throws java.sql.SQLException
public Object[] getDB2TrustedXAConnection(
  java.util.Properties properties)
  throws java.sql.SQLException
public Object[] getDB2TrustedXAConnection(
  org.ietf.jgss.GSSCredential gssCredential,
  java.util.Properties properties)
  throws java.sql.SQLException
```

Un servidor de aplicaciones que utiliza un ID de autorización del sistema utiliza este método para establecer una conexión fiable.

Las conexiones fiables están soportadas para:

- IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 para:
  - DB2 Database para Linux, UNIX y Windows Versión 9.5 o versiones posteriores
  - DB2 para z/OS Versión 9.1 o versiones posteriores
  - IBM Informix Versión 11.70 o versiones posteriores
- IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS Versión 9.1 o versiones posteriores

Los elementos siguientes se devuelven en Object[]:

- El primer elemento es una instancia de DB2TrustedXAConnection.
- El segundo elemento es una cookie exclusiva para la instancia de conexión generada de XA.

El primer formato getDB2TrustedXAConnection ofrece un ID de usuario y una contraseña. El segundo formato de getDB2TrustedXAConnection utiliza el ID de usuario y la contraseña del objeto DB2XADataSource. El tercer formato de getDB2TrustedXAConnection es para conexiones que utilizan la seguridad Kerberos.

Descripciones de parámetros:

user
El ID de autorización que se utiliza para establecer la conexión fiable.

contraseña
Contraseña del ID de autorización que se utiliza para establecer la conexión fiable.

gssCredential
Si la fuente de datos utiliza la seguridad Kerberos, especifica una credencial delegada que se pasa desde otro principal.

propiedades
Propiedades de la conexión.

getDB2TrustedPooledConnection

Formato:

```
public Object[] getDB2TrustedPooledConnection(java.util.Properties propiedades)
  throws java.sql.SQLException
```

Un servidor de aplicaciones que utiliza un ID de autorización del sistema utiliza este método para establecer una conexión fiable, utilizando el ID de usuario y la contraseña para el objeto DB2XADataSource.

Las conexiones fiables están soportadas para:

- IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 para:
  - DB2 Database para Linux, UNIX y Windows Versión 9.5 o versiones posteriores
  - DB2 para z/OS Versión 9.1 o versiones posteriores
  - IBM Informix Versión 11.70 o versiones posteriores
- IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS Versión 9.1 o versiones posteriores

Los elementos siguientes se devuelven en Object[]:

- El primer elemento es una instancia fiable de DB2TrustedPooledConnection.
- El segundo elemento es una cookie exclusiva para la instancia de conexión agrupada que se ha generado.

Descripciones de parámetros:

propiedades
Propiedades de la conexión.

getDB2XAConnection
Formatos:

```
public DB2XAConnection getDB2XAConnection(String user,
  String password,
  java.util.Properties properties)
  throws java.sql.SQLException
public DB2XAConnection getDB2XAConnection(
   org.ietf.jgss.GSSCredential gssCredential,
  java.util.Properties properties)
  throws java.sql.SQLException
```

Establece la conexión no fiable inicial en un entorno de agrupación heterogéneo.

El primer formato de getDB2PooledConnection ofrece un ID de usuario y una contraseña. El segundo formato de getDB2XAConnection es para conexiones que utilizan la seguridad Kerberos.

Descripciones de parámetros:

user
ID de autorización que se utiliza para establecer la conexión.

contraseña
Contraseña del ID de autorización que se utiliza para establecer la conexión.

gssCredential
Si la fuente de datos utiliza la seguridad Kerberos, especifica una credencial delegada que se pasa desde otro principal.

propiedades
Propiedades de la conexión.

Interfaz DB2Xml

La interfaz com.ibm.db2.jcc.DB2Xml se utiliza para declarar objetos Java para su utilización con el tipo de datos XML de DB2.

Métodos DB2Xml

El método siguiente sólo se define para IBM Data Server Driver para JDBC y SQLJ.

closeDB2Xml
Formato:

```
public void closeDB2Xml()
  throws SQLException
```

Libera los recursos que están asociados con un objeto com.ibm.jcc.DB2Xml.

getDB2AsciiStream
Formato:

```
public java.io.InputStream getDB2AsciiStream()
  throws SQLException
```

Recupera datos de un objeto DB2Xml, y convierte los datos en codificación US-ASCII.

getDB2BinaryStream

Formato:

```
public java.io.InputStream getDB2BinaryStream()
  throws SQLException
```

Recupera datos de un objeto DB2Xml como una corriente binaria. La codificación de caracteres de los bytes en la corriente binaria se define en la especificación XML 1.0.

getDB2Bytes

Formato:

```
public byte[] getDB2Bytes()
  throws SQLException
```

Recupera datos de un objeto DB2Xml como una matriz de bytes. La codificación de caracteres de los bytes se define en la especificación XML 1.0.

getDB2CharacterStream

Formato:

```
public java.io.Reader getDB2CharacterStream()
  throws SQLException
```

Recupera datos de un objeto DB2Xml como un objeto java.io.Reader.

getDB2String

Formato:

```
public String getDB2String()
  throws SQLException
```

Recupera datos de un objeto DB2Xml como valor de `String`.

getDB2XmlAsciiStream

Formato:

```
public InputStream getDB2XmlAsciiStream()
  throws SQLException
```

Recupera datos de un objeto DB2Xml, convierte los datos a codificación US-ASCII e incorpora una declaración XML con una especificación de codificación para US-ASCII en los datos devueltos.

getDB2XmlBinaryStream

Formato:

```
public java.io.InputStream getDB2XmlBinaryStream(String targetEncoding)
  throws SQLException
```

Recupera datos de un objeto DB2Xml como serie binaria, convierte los datos a *CodificaciónDestino* e incorpora una declaración XML con una especificación de codificación para *CodificaciónDestino* en los datos devueltos.

Parámetro:

targetEncoding
: Nombre de codificación válido que aparece en el registro de conjuntos de caracteres IANA. Los nombres de codificaciones que se pueden utilizar con el servidor DB2 están listados en "Correspondencias entre los CCSID y los nombres de codificaciones para datos de salida XML serializados".

getDB2XmlBytes

Formato:

```
public byte[] getDB2XmlBytes(String targetEncoding)
  throws SQLException
```

Recupera datos de un objeto DB2Xml como matriz binaria, convierte los datos a *CodificaciónDestino* e incorpora una declaración XML con una especificación de codificación para *CodificaciónDestino* en los datos devueltos.

Parámetro:

targetEncoding
Nombre de codificación válido que aparece en el registro de conjuntos de caracteres IANA. Los nombres de codificaciones que se pueden utilizar con el servidor DB2 están listados en "Correspondencias entre los CCSID y los nombres de codificaciones para datos de salida XML serializados".

getDB2XmlCharacterStream

Formato:

```
public java.io.Reader getDB2XmlCharacterStream()
  throws SQLException
```

Recupera datos de un objeto DB2Xml como objeto java.io.Reader, convierte los datos en codificación ISO-10646-UCS-2 e incorpora una declaración XML con una especificación de codificación para ISO-10646-UCS-2 en los datos devueltos.

getDB2XmlString

Formato:

```
public String getDB2XmlString()
  throws SQLException
```

Recupera datos de un objeto DB2Xml como objeto `String`, convierte los datos en codificación ISO-10646-UCS-2 e incorpora una declaración XML con una especificación de codificación para ISO-10646-UCS-2 en los datos devueltos.

isDB2XmlClosed

Formato:

```
public boolean isDB2XmlClosed()
  throws SQLException
```

Indica si se ha cerrado un objeto com.ibm.jcc.DB2Xml.

Clase DBTimestamp

La clase com.ibm.db2.jcc.DBTimestamp se puede utilizar para crear objetos de indicación de fecha y hora con una precisión incluso de picosegundos e información del huso horario. Esta clase se destina principalmente al soporte del tipo de datos de SQL TIMESTAMP WITH TIME ZONE, que solamente se soporta en DB2 para z/OS.

La clase com.ibm.db2.jcc.DBTimestamp es una subclase de la clase java.sql.Timestamp. Por consiguiente, un objeto com.ibm.db2.jcc.DBTimestamp se puede utilizar con cualquier método que funcione normalmente en un objeto java.sql.Timestamp, o que toman un objeto java.sql.Timestamp como argumento.

IBM Data Server Driver para JDBC y SQLJ devuelve un objeto DBTimestamp para todos los métodos JDBC que devuelven información de indicación de fecha y hora, como ResultSet.getTimestamp o CallableStatement.getTimestamp.

Constructor DBTimestamp

El constructor siguiente sólo se define para IBM Data Server Driver para JDBC y SQLJ.

DBTimestamp

Formatos:

```
public DBTimestamp(long hora,
  java.util.Calendar calendario)
  throws java.sql.SQLException
public DBTimestamp(long hora)
  throws java.sql.SQLException
public DBTimestamp(java.sql.Timestamp IndicaciónFechaHora)
  throws java.sql.SQLException
public DBTimestamp(java.sql.Timestamp IndicaciónFechaHora,
  java.util.Calendar calendario)
  throws java.sql.SQLException
```

Crea un objeto DBTimestamp.

Descripciones de parámetros:

hora
Número de milisegundos desde el 1 de enero de 1970.

IndicaciónFechaHora
Un valor Timestamp con una precisión incluso de picosegundos.

calendario
El valor Calendar que proporciona el huso horario.

Métodos DBTimestamp

getPicos

Formatos:

```
public long getPicos()
```

Devuelve el componente de segundos fraccionarios de un valor DBTimestamp.

getTimeZone

Formatos:

```
public java.util.TimeZone getTimeZone()
```

Devuelve el componente de huso horario de un valor DBTimestamp.

setPicos

Formato:

```
public void setPicos(long p)
  throws SQLException
```

Asigna el valor dado al componente de segundos fraccionarios de un valor DBTimestamp.

Descripciones de parámetros:

p Un valor entre 0 y 999999999999, incluidos, que es el componente de secciones fraccionarias de un valor DBTimestamp.

setTimeZone

Formato:

```
public void setTimeZone(java.util.TimeZone huso_horario)
  throws SQLException
```

Asigna el valor dado al componente de huso horario de un valor DBTimestamp.

Descripciones de parámetros:

timeZone
El componente de huso horario de un valor DBTimestamp.

valueOfDBString
Formato:

```
public static DBTimestamp valueOfDBString(String s)
  throws java.lang.IllegalArgumentException
```

Crea un valor DBTimestamp a partir de la representación de serie de un valor de indicación de fecha y hora.

Descripciones de parámetros:

s La representación de serie de un valor de indicación de fecha y hora. El valor debe tener uno de los formatos siguientes:

aaaa-mm-dd.hh.mm.ss[.fffffffffff]-th:tm
aaaa-mm-dd hh:mm:ss[.fffffffffff]-th:tm
aaaa-mm-dd.hh.mm.ss[.fffffffffff]
aaaa-mm-dd hh:mm:ss[.fffffffffff]

- *aaaa* es un año.
- *mm* es un mes.
- *dd* es un día.
- *hh* es la hora.
- *mm* indica los minutos.
- *ss* indica los segundos.
- *[.fffffffffff]* indica entre una y 12 fracciones de segundos opcionales.
- *th* es el componente de hora de un huso horario.
- *tm* es el componente de minutos de un huso horario.

toDBString
Formato:

```
public String toDBString(boolean incluirHusoHorario)
```

Devuelve una representación de serie de un objeto DBTimestamp.

El valor devuelto tiene uno de los formatos siguientes:

aaaa-mm-dd.hh.mm.ss[.fffffffffff]-th:tm
aaaa-mm-dd.hh.mm.ss[.fffffffffff]

Descripción de parámetros:

incluirHusoHorario
Especifica si se incluye el huso horario (*-th:tm*) en la serie devuelta.

Diferencias de JDBC entre versiones de IBM Data Server Driver para JDBC y SQLJ

Antes de poder actualizar sus aplicaciones JDBC desde versiones más antiguas a versiones más nuevas de IBM Data Server Driver para JDBC y SQLJ, es necesario que comprenda las diferencias entre esos controladores.

Métodos admitidos

Para obtener una lista de los métodos a los que IBM Data Server Driver para JDBC y SQLJ da soporte, consulte la información sobre el soporte del controlador para las API JDBC.

Utilización de la modalidad continua progresiva por los controladores JDBC

Para IBM Data Server Driver para JDBC y SQLJ, Versión 3.50 y posterior, la modalidad continua progresiva, también conocida como formato de datos dinámicos, es el comportamiento por omisión para recuperar objetos LOB en las conexiones con DB2 Database para Linux, UNIX y Windows Versión 9.5 y posterior.

La modalidad continua progresiva se puede utilizar en IBM Data Server Driver para JDBC y SQLJ Versión 3.1 y posterior, pero para IBM Data Server Driver para JDBC y SQLJ Versión 3.2 y posterior, la modalidad continua progresiva es el comportamiento por omisión para recuperar objetos LOB y XML en las conexiones con DB2 para z/OS Versión 9.1 y posterior.

Las versiones anteriores de IBM Data Server Driver para JDBC y SQLJ no daban soporte a la modalidad continua progresiva.

Importante: Con la modalidad continua progresiva, al recuperar un valor LOB o XML de un ResultSet en una variable de la aplicación, podrá manipular el contenido de dicha variable de la aplicación hasta que mueva el cursor o lo cierre en ResultSet. Tras esta acción, ya no podrá acceder al contenido de la variable de la aplicación. Si lleva a cabo acciones en el LOB de la variable de la aplicación, recibirá una excepción SQLException. Por ejemplo, suponga que la modalidad continua progresiva está habilitada y ejecuta sentencias del tipo siguiente:

```
...
ResultSet rs = stmt.executeQuery("SELECT CLOBCOL FROM MY_TABLE");
rs.next();                          // Recuperar la primera fila de ResultSet
Clob clobFromRow1  = rs.getClob(1);
                          // Colocar el CLOB de la primera columna de
                          // la primera fila en una variable de aplicación
String substr1Clob = clobFromRow1.getSubString(1,50);
                          // Recuperar los primeros 50 bytes de CLOB
rs.next();                          // Mover el cursor hasta la fila siguiente.
                          // clobFromRow1 ya no se encuentra disponible.
// String substr2Clob = clobFromRow1.getSubString(51,100);
                          // Esta sentencia daría lugar a una excepción
                                    // SQLException
Clob clobFromRow2  = rs.getClob(1);
                          // Colocar el CLOB de la primera columna de
                          // la segunda fila en una variable de aplicación
rs.close();                         // Cerrar el ResultSet.
                          // clobFromRow2 tampoco se encuentra disponible.
```

Una vez que haya ejecutado rs.next() para colocar el cursor en la segunda fila de ResultSet, el valor CLOB de clobFromRow1 dejará de estar disponible. De forma similar, una vez que haya ejecutado rs.close() para cerrar el ResultSet, los valores de clobFromRow1 y clobFromRow2 dejarán de estar disponibles.

Para evitar errores debidos a este comportamiento cambiado, es necesario que emprenda una de las acciones siguientes:

- Modifique sus aplicaciones.

Las aplicaciones que insertan datos LOB en variables de aplicación pueden manejar los datos de esas variables de aplicación solamente hasta que muevan o cierren los cursores que se utilizaron para recuperar los datos.

- Inhabilite la modalidad continua progresiva estableciendo la propiedad `progressiveStreaming` en `DB2BaseDataSource.NO` (2).

Valores de ResultSetMetaData para IBM Data Server Driver para JDBC y SQLJ versión 4.0 y posterior

Para IBM Data Server Driver para JDBC y SQLJ versión 4.0 y posterior, el comportamiento por omisión de ResultSetMetaData.getColumnName y `ResultSetMetaData.getColumnLabel` difiere del comportamiento por omisión para controladores JDBC de versiones anteriores.

Si necesita utilizar IBM Data Server Driver para JDBC y SQLJ versión 4.0 o posterior, pero sus aplicaciones necesitan devolver los valores de ResultSetMetaData.getColumnName y `ResultSetMetaData.getColumnLabel` que fueron devueltos para controladores JDBC más antiguos, puede establecer la propiedad useJDBC4ColumnNameAndLabelSemantics Connection y `DataSource` en `DB2BaseDataSource.NO` (2).

Las actualizaciones por lotes con claves generadas automáticamente producen resultados diferentes con versiones diferentes del controlador

En IBM Data Server Driver para JDBC y SQLJ versión 3.52 o posterior, se puede preparar una sentencia de SQL para recuperar claves generadas automáticamente.

Cuando se utiliza IBM Data Server Driver para JDBC y SQLJ versión 3.50 o 3.51, el preparar una sentencia de SQL para recuperar claves generadas automáticamente y utilizar el objeto PreparedStatement para actualizaciones por lotes produce una excepción SQLException.

Las versiones de IBM Data Server Driver para JDBC y SQLJ anteriores a la Versión 3.50 no emiten un SQLException cuando una aplicación invoca el método addBatch o executeBatch para un objeto PreparedStatement que está preparado para devolver claves generadas automáticamente. Pero el objeto PreparedStatement no devuelve claves generadas automáticamente.

Las actualizaciones por lotes de datos en servidores DB2 para z/OS generan resultados diferentes en versiones de controlador distintas

Tras invocar satisfactoriamente una sentencia executeBatch, IBM Data Server Driver para JDBC y SQLJ devuelve una matriz. El objetivo de la matriz es indicar el número de filas afectadas por cada sentencia de SQL que se ejecuta en el proceso por lotes.

Si se cumplen las condiciones siguientes, IBM Data Server Driver para JDBC y SQLJ devuelve `Statement.SUCCESS_NO_INFO` (-2) en los elementos de matriz:

- La aplicación está conectada con un subsistema que está en DB2 para z/OS Versión 8 en modalidad de función nueva, o posteriores.
- La aplicación utiliza la Versión 3.1 o posteriores de IBM Data Server Driver para JDBC y SQLJ.

- IBM Data Server Driver para JDBC y SQLJ utiliza operaciones INSERT de varias filas para ejecutar las actualizaciones por lotes.

Esto se debe a que, al realizar una operación INSERT de varias filas, el servidor de bases de datos ejecuta todo el proceso por lotes como una única operación, de forma que no devuelve resultados para sentencias de SQL individuales.

Si está utilizando una versión anterior de IBM Data Server Driver para JDBC y SQLJ o está conectado con una fuente de datos distinta de DB2 para z/OS Versión 8 o posterior, los elementos de matriz contienen el número de filas afectadas por cada sentencia de SQL.

Las actualizaciones y supresiones por lotes de datos en servidores DB2 para z/OS tienen limitaciones de tamaño diferentes en versiones de controlador distintas

Antes de IBM Data Server Driver para JDBC y SQLJ versión 3.59 ó 4.9, se generaba una excepción DisconnectException con el código de error -4499 para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 para DB2 para z/OS si el tamaño de una actualización o una supresión por lotes superaba los 32 KB. A partir de la versión 3.59 ó 4.9, esta restricción ya no existe y la excepción ya no se genera.

Valor inicial del registro especial CURRENT CLIENT_ACCTNG

Para una aplicación JDBC o SQLJ que se ejecuta bajo IBM Data Server Driver para JDBC y SQLJ versión 2.6 o posterior, y se utiliza la conectividad de tipo 4, el valor inicial del registro especial CURRENT CLIENT_ACCTNG de DB2 para z/OS resulta de concatenar la versión de DB2 para z/OS y el valor de la propiedad clientWorkStation. Para cualquier otro controlador JDBC, versión y conectividad, el valor inicial no está establecido.

Propiedades que controlan la utilización de la sentencia FETCH de varias filas

Con anterioridad a la versión 3.7 y 3.51 de IBM Data Server Driver para JDBC y SQLJ, la sentencia FETCH de varias filas se habilitaba e inhabilitaba mediante la propiedad useRowsetCursor, y solamente se podía utilizar para cursores desplazables, y para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 con DB2 para z/OS. A partir de la versión 3.7 y 3.51:

- Para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS, IBM Data Server Driver para JDBC y SQLJ solo utiliza la propiedad enableRowsetSupport para determinar si debe utilizarse la sentencia FETCH de varias filas para los cursores desplazables o de solo avance.
- Para IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4 con DB2 para z/OS o DB2 Database para Linux, UNIX y Windows, o IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 Database para Linux, UNIX y Windows, IBM Data Server Driver para JDBC y SQLJ utiliza la propiedad enableRowsetSupport para determinar si debe utilizar la sentencia FETCH de varias filas para cursores desplazables, si enableRowsetSupport está definido. Si enableRowsetSupport no está definido, el controlador utiliza la propiedad useRowsetCursor para determinar si debe utilizar la sentencia FETCH de varias filas.

Actualizaciones y supresiones de posición de JDBC 1 y FETCH de varias filas

Con anterioridad a la versión 3.7 y 3.51 de IBM Data Server Driver para JDBC y SQLJ, la sentencia FETCH de varias filas sobre tablas DB2 para z/OS estaba controlada por la propiedad useRowsetCursor. Si una aplicación contenía operaciones de actualización o supresión posicionada de JDBC 1, y estaba habilitado el soporte de operaciones FETCH de varias filas, IBM Data Server Driver para JDBC y SQLJ permitía las operaciones de actualización o supresión, pero podían producirse actualizaciones o supresiones no esperadas.

A partir de la versión 3.7 y 3.51 de IBM Data Server Driver para JDBC y SQLJ, la propiedad enableRowsetSupport habilita o inhabilita el uso de sentencias FETCH de varias filas sobre tablas de DB2 para z/OS o tablas de DB2 Database para Linux, UNIX y Windows. La propiedad enableRowsetSupport tiene preferencia sobre la propiedad useRowsetCursor. Si el uso de sentencias FETCH de varias filas se habilita mediante la propiedad enableRowsetSupport, y una aplicación contiene una operación de actualización o supresión de posición de JDBC 1, IBM Data Server Driver para JDBC y SQLJ emite una excepción SQLException.

Formatos válidos de prepareStatement para recuperación de claves generadas automáticamente desde una vista de DB2 para z/OS

A partir de la versión 3.57 o la versión 4.7 de IBM Data Server Driver para JDBC y SQLJ, si está insertando datos en una vista de un servidor de datos DB2 para z/OS y desea recuperar claves generadas automáticamente, tendrá que usar uno de los métodos siguientes para preparar la sentencia de SQL que inserta filas en la vista:

```
Connection.prepareStatement(sentencia-sql, String [] nombresColumna);
Connection.prepareStatement(sentencia-sql, int [] Índicescolumna);
Statement.executeUpdate(sentencia-sql, String [] nombresColumna);
Statement.executeUpdate(sentencia-sql, int [] Índicescolumna);
```

Pérdida de datos para las actualizaciones de columna TIMESTAMP(p) mediante setString

Si utiliza una llamada setString para pasar un valor de entrada a una columna TIMESTAMP(*p*), es posible enviar un valor con una precisión mayor que nueve a la columna.

Con anterioridad a la versión 3.59 o a la versión 4.9 de IBM Data Server Driver para JDBC y SQLJ, podía producirse pérdida de datos si la propiedad sendDataAsIs se establecía en `false` y la precisión del valor de entrada era mayor que nueve.

A partir de la versión 3.59 y de la versión 4.9 de IBM Data Server Driver para JDBC y SQLJ, la pérdida de datos no se produce si la columna TIMESTAMP(*p*) tiene el tamaño suficiente para alojar el valor de entrada.

Cambio en el nombre de columna de conjunto de resultados de getColumns

En la versión 4.12 o anterior de IBM Data Server Driver para JDBC y SQLJ, el método DatabaseMetaData.getColumns devolvía un conjunto de resultados que contenía una columna denominada SCOPE_CATLOG. En la versión 4.13 o posterior de IBM Data Server Driver para JDBC y SQLJ, el nombre de esa columna

es SCOPE_CATALOG. Si desea que IBM Data Server Driver para JDBC y SQLJ siga utilizando el nombre de columna SCOPE_CATLOG, establezca la propiedad de DataSource o Connection useJDBC41DefinitionForGetColumns en DB2BaseDataSource.NO (2).

Cambio en el nombre de columna de conjunto de resultados de getColumns

En la versión 4.12 o anterior de IBM Data Server Driver para JDBC y SQLJ, el método DatabaseMetaData.getColumns devolvía un conjunto de resultados que contenía una columna denominada SCOPE_CATLOG. En la versión 4.13 o posterior de IBM Data Server Driver para JDBC y SQLJ, el nombre de esa columna es SCOPE_CATALOG. Si desea que IBM Data Server Driver para JDBC y SQLJ siga utilizando el nombre de columna SCOPE_CATLOG, establezca la propiedad de DataSource o Connection useJDBC41DefinitionForGetColumns en DB2BaseDataSource.NO (2).

Cambios en los valores por omisión para las propiedades de configuración global db2.jcc.maxRefreshInterval, db2.jcc.maxTransportObjects y db2.jcc.maxTransportObjectWaitTime

Los valores por omisión para las propiedades de configuración global db2.jcc.maxRefreshInterval, db2.jcc.maxTransportObjects y db2.jcc.maxTransportObjectWaitTime cambian en la Versión 3.63 y 4.13 de IBM Data Server Driver para JDBC y SQLJ. En la tabla siguiente se enumeran los valores por omisión anteriores y los nuevos.

Propiedad de configuración	Valor por omisión antes de las versiones 3.63 y 4.13	Valor por omisión para las versiones 3.63, 4.13 o posteriores
db2.jcc.maxRefreshInterval	30 segundos	10 segundos
db2.jcc.maxTransportObjects	-1 (ilimitado)	1000
db2.jcc.maxTransportObjectWaitTime	-1 (ilimitado)	1 segundo

Cambios en los valores por omisión para las propiedades maxRetriesForClientReroute, maxTransportObjects y retryIntervalForClientReroute de Connection y DataSource

Los valores por omisión para las propiedades maxRetriesForClientReroute, maxTransportObjects y retryIntervalForClientReroute de Connection y DataSource cambian en la versión 3.63 y 4.13 de IBM Data Server Driver para JDBC y SQLJ. En la tabla siguiente se enumeran los valores por omisión anteriores y los nuevos.

Propiedad de Connection y DataSource	Valor por omisión antes de las versiones 3.63 y 4.13	Valor por omisión para las versiones 3.63, 4.13 o posteriores
maxRetriesForClientReroute	Si maxRetriesForClientReroute y retryIntervalForClientReroute no se han establecido, la conexión se reintenta durante 10 minutos, con un tiempo de espera entre reintentos que aumenta a medida que se incrementa el tiempo transcurrido desde el primer reintento.	Si maxRetriesForClientReroute y retryIntervalForClientReroute no se han establecido, la propiedad enableSysplexWLB está establecida en `true` y el servidor de datos es DB2 para z/OS, el valor por omisión es 5. De lo contrario, el valor por omisión es el mismo que para las versiones anteriores del controlador.
maxTransportObjects	-1 (ilimitado)	1000
retryIntervalForClientReroute	Si maxRetriesForClientReroute y retryIntervalForClientReroute no se han establecido, la conexión se reintenta durante 10 minutos, con un tiempo de espera entre reintentos que aumenta a medida que se incrementa el tiempo transcurrido desde el primer reintento.	Si maxRetriesForClientReroute y retryIntervalForClientReroute no se han establecido, la propiedad enableSysplexWLB está establecida en `true` y el servidor de datos es DB2 para z/OS, el valor por omisión es 0 segundos. De lo contrario, el valor por omisión es el mismo que para las versiones anteriores del controlador.

Cambios en los valores por omisión para las propiedades de información del cliente en DB2 para z/OS

Los valores por omisión para las propiedades de información del cliente en IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 2 en DB2 para z/OS cambian en la Versión 3.64 y 4.14 de IBM Data Server Driver para JDBC y SQLJ. En la tabla siguiente se enumeran los valores por omisión anteriores y los nuevos.

Propiedad de información del cliente	Valor por omisión antes de las versiones 3.64 y 4.14	Valor por omisión para las versiones 3.64, 4.14 o posteriores
ApplicationName	Serie de caracteres vacía	La serie "db2jcc_application"
ClientAccountingInformation	Serie de caracteres vacía	Serie de caracteres vacía
ClientHostname	Serie de caracteres vacía	La serie "RRSAF"
ClientUser	Serie de caracteres vacía	El ID de usuario que se especificó para la conexión. Si no se ha especificado ningún ID de usuario, se utiliza el ID de usuario de RACF.

Cambios en el comportamiento de la propiedad xmlFormat

A partir de la versión 4.15 de IBM Data Server Driver para JDBC y SQLJ, la propiedad xmlFormat de Connection y DataSource solo se aplica a la recuperación de datos XML, en lugar de aplicarse a la actualización y recuperación de datos XML. Asimismo, el comportamiento por omisión ha cambiado por la recuperación de datos XML en formato XML de texto, con independencia de si el servidor de datos da soporte o no al formato XML binario.

Para la actualización de datos en las columnas XML, xmlFormat no tiene ningún efecto. Si los datos de entrada son datos XML binarios, y el servidor de datos no da soporte a los datos XML binarios, los datos de entrada se convierten en datos XML de texto. De otro modo, no tiene lugar ninguna conversión.

Ejemplos de valores de ResultSetMetaData.getColumnName y ResultSetMetaData.getColumnLabel

Para IBM Data Server Driver para JDBC y SQLJ versión 4.0 y posterior, el comportamiento por omisión de ResultSetMetaData.getColumnName y `ResultSetMetaData.getColumnLabel` difiere del comportamiento por omisión para controladores JDBC de versiones anteriores. Puede utilizar la propiedad useJDBC4ColumnNameAndLabelSemantics para cambiar este comportamiento.

Los ejemplos siguientes muestran los valores que se devuelven para IBM Data Server Driver para JDBC y SQLJ Versión 4.0, y para controladores JDBC anteriores, cuando la propiedad useJDBC4ColumnNameAndLabelSemantics no está establecida.

Todas las consultas utilizan una tabla que está definida de esta manera:

```
CREATE TABLE MYTABLE(INTCOL INT)
```

Ejemplo: la consulta siguiente contiene una cláusula AS, que define una etiqueta para una columna del conjunto de resultados:

```
SELECT MYCOL AS MYLABEL FROM MYTABLE
```

La tabla siguiente lista los valores de ResultSetMetaData.getColumnName y ResultSetMetaData.getColumnName que se devuelven para la consulta:

Tabla 114. Valores de ResultSetMetaData.getColumnName y ResultSetMetaData.getColumnName devueltos para una consulta con una cláusula AS en las versiones de IBM Data Server Driver para JDBC y SQLJ anteriores y posteriores a la versión 4.0

	Comportamiento para IBM Data Server Driver para JDBC y SQLJ anterior a la versión 4.0		**Comportamiento para IBM Data Server Driver para JDBC y SQLJ versión 4.0 y posterior**	
Fuente de datos de destino	**Valor de getColumnName**	**Valor de getColumnLabel**	**Valor de getColumnName**	**Valor de getColumnLabel**
DB2 Database para Linux, UNIX y Windows	MYLABEL	MYLABEL	MYCOL	MYLABEL
IBM Informix	MYLABEL	MYLABEL	MYCOL	MYLABEL

Tabla 114. Valores de ResultSetMetaData.getColumnName y ResultSetMetaData.getColumnName devueltos para una consulta con una cláusula AS en las versiones de IBM Data Server Driver para JDBC y SQLJ anteriores y posteriores a la versión 4.0 (continuación)

	Comportamiento para IBM Data Server Driver para JDBC y SQLJ anterior a la versión 4.0		Comportamiento para IBM Data Server Driver para JDBC y SQLJ versión 4.0 y posterior	
Fuente de datos de destino	**Valor de getColumnName**	**Valor de getColumnLabel**	**Valor de getColumnName**	**Valor de getColumnLabel**
DB2 para z/OS Versión 8 o posteriores y DB2 UDB para iSeries V5R3 y posteriores	MYLABEL	MYLABEL	MYCOL	MYLABEL
DB2 para z/OS Versión 7 y DB2 UDB para iSeries V5R2	MYLABEL	MYLABEL	MYLABEL	MYLABEL

Ejemplo: la consulta siguiente no contiene ninguna cláusula AS:

```
SELECT MYCOL FROM MYTABLE
```

Los métodos ResultSetMetaData.getColumnName y ResultSetMetaData.getColumnLabel de la consulta devuelven MYCOL, cualquiera que sea la fuente de datos de destino.

Ejemplo: en una fuente de datos DB2 para z/OS o DB2 para i, se utiliza una sentencia LABEL ON para definir una etiqueta para una columna:

```
LABEL ON COLUMN MYTABLE.MYCOL IS 'LABELONCOL'
```

La consulta siguiente contiene una cláusula AS, que define una etiqueta para una columna de ResultSet (conjunto de resultados):

```
SELECT MYCOL AS MYLABEL FROM MYTABLE
```

La tabla siguiente lista los valores de ResultSetMetaData.getColumnName y ResultSetMetaData.getColumnName que se devuelven para la consulta.

Tabla 115. Valores de ResultSetMetaData.getColumnName y ResultSetMetaData.getColumnName devueltos para una columna de tabla con una sentencia LABEL ON en una consulta con una cláusula AS en las versiones de IBM Data Server Driver para JDBC y SQLJ anteriores y posteriores a la versión 4.0

	Comportamiento para IBM Data Server Driver para JDBC y SQLJ anterior a la versión 4.0		Comportamiento para IBM Data Server Driver para JDBC y SQLJ versión 4.0 y posterior	
Fuente de datos de destino	**Valor de getColumnName**	**Valor de getColumnLabel**	**Valor de getColumnName**	**Valor de getColumnLabel**
DB2 para z/OS Versión 8 o posteriores y DB2 UDB para iSeries V5R3 y posteriores	MYLABEL	LABELONCOL	MYCOL	MYLABEL
DB2 para z/OS Versión 7 y DB2 UDB para iSeries V5R2	MYLABEL	LABELONCOL	MYCOL	LABELONCOL

Ejemplo: en una fuente de datos DB2 para z/OS o DB2 para i, se utiliza una sentencia LABEL ON para definir una etiqueta para una columna:

```
LABEL ON COLUMN MYTABLE.MYCOL IS 'LABELONCOL'
```

La consulta siguiente no contiene ninguna cláusula AS:

```
SELECT MYCOL FROM MYTABLE
```

La tabla siguiente lista los valores de ResultSetMetaData.getColumnName y ResultSetMetaData.getColumnName que se devuelven para la consulta.

Tabla 116. ResultSetMetaData.getColumnName y ResultSetMetaData.getColumnName antes y después de IBM Data Server Driver para JDBC y SQLJ versión 4.0 para una columna de tabla con una sentencia LABEL ON en una consulta sin ninguna AS CLAUSE

	Comportamiento para IBM Data Server Driver para JDBC y SQLJ anterior a la versión 4.0		**Comportamiento para IBM Data Server Driver para JDBC y SQLJ versión 4.0**	
Fuente de datos de destino	**Valor de getColumnName**	**Valor de getColumnLabel**	**Valor de getColumnName**	**Valor de getColumnLabel**
DB2 para z/OS versión 8 o posterior, y DB2 UDB para i5/OS V5R3 y posteriores	MYCOL	LABELONCOL	MYCOL	MYCOL
DB2 para z/OS versión 7 y DB2 UDB para i5/OS V5R2	MYCOL	LABELONCOL	MYLABEL	LABELONCOL

Diferencias del SDK de Java que afectan al IBM Data Server Driver para JDBC y SQLJ

Las diferencias de comportamiento entre las versiones del SDK de Java pueden producir variaciones en los resultados que obtiene al ejecutar programas bajo IBM Data Server Driver para JDBC y SQLJ.

Valores recuperados para caracteres de sustitución DBCS

Cuando recupera un carácter de sustitución DBCS, como X'FCFC' en la página de códigos Cp943, de una tabla de base de datos, el valor devuelto difiere en función de si está utilizando un SDK de IBM SDK para Java o un SDK para Java de Oracle.

Para un SDK para Java de Oracle, el carácter de sustitución se recupera como U+0000. Para un SDK de Java de IBM, el carácter de sustitución se recupera como X'FFFD'.

Páginas de códigos permitidas

Los SDK de IBM para Java dan soporte más páginas de códigos DBCS que los SDK para Java de Oracle. Por tanto, si recibe errores porque existen páginas de códigos no soportadas con un SDK para Java de Oracle, intente utilizar un SDK de IBM SDK para Java.

Requisito de cifrado del SDK de IBM para Java

Los SDK de IBM para Java dan soporte al cifrado de 256 bits, pero los SDK para Java de Oracle no disponen de este soporte. Por tanto, si utiliza cualquiera de los mecanismos de seguridad de IBM Data Server Driver para JDBC y SQLJ en los que intervenga el cifrado, es necesario que utilice un SDK de IBM para Java.

Soporte para la supervisión del sistema

El soporte para la supervisión del sistema en IBM Data Server Driver para JDBC y SQLJ incluye la recogida del tiempo de controlador básico y del tiempo de entrada/salida de red. La obtención de esta información requiere funciones que existen en cualquier SDK para Java Versión 5 o posterior. Sin embargo, el SDK de IBM para Java Versión 1.4.2 también permite la recogida del tiempo de controlador básico y del tiempo de entrada/salida de red. Si utiliza el SDK de IBM para Java Versión 1.4.2, el tiempo de controlador básico y el tiempo de entrada/salida de red se redondean hasta el microsegundo más cercano. Si utiliza un SDK para Java Versión 5 o posterior, el tiempo de controlador básico y el tiempo de entrada/salida de red se redondean hasta el nanosegundo más cercano.

Códigos de error emitidos por IBM Data Server Driver para JDBC y SQLJ

Los códigos de error comprendidos dentro de los rangos +4200 a +4299, +4450 a +4499, -4200 a -4299, y -4450 a -4499 están reservados para IBM Data Server Driver para JDBC y SQLJ.

Cuando invoca el método SQLException.getMessage después de producirse un error de IBM Data Server Driver para JDBC y SQLJ, el método devuelve una serie de caracteres que incluye lo siguiente:
- Indicación si la conexión es de tipo 2 o tipo 4
- Información de diagnóstico para el centro de soporte de software de IBM
- Nivel del controlador
- Un mensaje explicativo
- El código de error
- El SQLSTATE

Por ejemplo:

```
[jcc][t4][20128][12071][3.50.54] Invalid queryBlockSize specified: 1,048,576,012.
Using default query block size of 32,767.  ERRORCODE=0, SQLSTATE=
```

Actualmente, IBM Data Server Driver para JDBC y SQLJ emite los códigos de error siguientes:

Tabla 117. Códigos de error emitidos por IBM Data Server Driver para JDBC y SQLJ

Código de error	Texto del mensaje y explicación	SQLSTATE
+4204	Se han encontrado errores y se han tolerado según lo especificado en la cláusula RETURN DATA UNTIL. **Explicación:** Los errores tolerados incluyen errores de autorización, autentificación y conexión federada. Este aviso se aplica solamente a las conexiones con los servidores DB2 Database para Linux, UNIX y Windows. Se emite solamente cuando una operación del cursor, como una llamada ResultSet.next o ResultSet.previous, devuelve `false`.	02506

Tabla 117. Códigos de error emitidos por IBM Data Server Driver para JDBC y SQLJ (continuación)

Código de error	Texto del mensaje y explicación	SQLSTATE
+4222	*texto-de-getMessage* **Explicación:** se produjo una condición de aviso al conectarse con la fuente de datos. **Respuesta de usuario:** llamar a SQLException.getMessage para recuperar información específica sobre el problema.	
+4223	*texto-de-getMessage* **Explicación:** se produjo una condición de aviso durante la inicialización. **Respuesta de usuario:** llamar a SQLException.getMessage para recuperar información específica sobre el problema.	
+4225	*texto-de-getMessage* **Explicación:** se produjo una condición de aviso al enviar datos a un servidor o al recibirlos de un servidor. **Respuesta de usuario:** llamar a SQLException.getMessage para recuperar información específica sobre el problema.	
+4226	*texto-de-getMessage* **Explicación:** se produjo una condición de aviso durante la personalización o la vinculación. **Respuesta de usuario:** llamar a SQLException.getMessage para recuperar información específica sobre el problema.	
+4228	*texto-de-getMessage* **Explicación:** se produjo una condición de aviso que no se adecua en otra categoría. **Respuesta de usuario:** llamar a SQLException.getMessage para recuperar información específica sobre el problema.	
+4450	No se soporta la característica: *nombre-característica*.	
+4460	*texto-de-getMessage* **Explicación:** el valor especificado no es una opción válida. **Respuesta de usuario:** llamar a SQLException.getMessage para recuperar información específica sobre el problema.	
+4461	*texto-de-getMessage* **Explicación:** el valor especificado no es válido o se encuentra fuera del intervalo esperado. **Respuesta de usuario:** llamar a SQLException.getMessage para recuperar información específica sobre el problema.	
+4462	*texto-de-getMessage* **Explicación:** falta un valor necesario. **Respuesta de usuario:** llamar a SQLException.getMessage para recuperar información específica sobre el problema.	

Tabla 117. Códigos de error emitidos por IBM Data Server Driver para JDBC y SQLJ (continuación)

Código de error	Texto del mensaje y explicación	SQLSTATE
+4470	*texto-de-getMessage* **Explicación:** la operación solicitada no se puede realizar debido a que el recurso de destino está cerrado. **Respuesta de usuario:** llamar a SQLException.getMessage para recuperar información específica sobre el problema.	
+4471	*texto-de-getMessage* **Explicación:** la operación solicitada no se puede realizar debido a que se está utilizando el recurso de destino. **Respuesta de usuario:** llamar a SQLException.getMessage para recuperar información específica sobre el problema.	
+4472	*texto-de-getMessage* **Explicación:** la operación solicitada no se puede realizar debido a que el recurso de destino no está disponible. **Respuesta de usuario:** llamar a SQLException.getMessage para recuperar información específica sobre el problema.	
+4474	*texto-de-getMessage* **Explicación:** la operación solicitada no se puede realizar debido a que el recurso de destino no se puede cambiar. **Respuesta de usuario:** llamar a SQLException.getMessage para recuperar información específica sobre el problema.	
-1224	El gestor de bases de datos no puede aceptar peticiones nuevas, ha interrumpido todas las peticiones en proceso o ha interrumpido la petición especificada debido a un error o una interrupción forzada. **Explicación:** Para las conexiones con servidores de datos DB2 Database para Linux, UNIX y Windows, consulte el apartado para obtener detalles. Para las conexiones con servidores de datos DB2 para z/OS, este error indica que la hebra del servidor DB2 para z/OS que se asocia a la aplicación ha finalizado de forma anómala. Deben recopilarse los diagnósticos de servidor relacionados con la aplicación. La aplicación puede identificarse mediante su ID de aplicación exclusivo. DB2 para z/OS externaliza el ID de aplicación en los mensajes y rastreos como símbolo de correlación de conexión (CRRTKN) e ID de unidad lógica de trabajo (LUWID). En la consola de z/OS se genera el mensaje DSNL027I cuando una hebra de DB2 para z/OS finaliza de forma anómala. El mensaje DSNL027I proporciona un código de razón para la anomalía. En la mayoría de los casos, DB2 para z/OS genera un vuelco de SVC de z/OS, necesario para solucionar el problema.	58009
-4200	Operación no válida: Se ha llamado una operación COMMIT o ROLLBACK no válida en un entorno XA durante una transacción global. **Explicación:** Una aplicación perteneciente a una transacción global en un entorno XA ha emitido una operación de confirmación o retrotracción. Una operación de confirmación o retrotracción en una transacción global no es válida.	2D521

Tabla 117. Códigos de error emitidos por IBM Data Server Driver para JDBC y SQLJ (continuación)

Código de error	Texto del mensaje y explicación	SQLSTATE
-4201	Operación no válida: no se permite setAutoCommit(true) durante la transacción global. **Explicación:** Una aplicación perteneciente a una transacción global en un entorno XA ha ejecutado la sentencia setAutoCommit(true). La emisión de setAutoCommit(true) en una transacción global no es válida.	2D521
-4203	Error al ejecutar la *función*. El servidor ha devuelto *rc*. **Explicación:** Se ha producido un error en una conexión XA durante la ejecución de una sentencia de SQL. Para la optimización de la red, IBM Data Server Driver para JDBC y SQLJ retarda algunos flujos XA hasta que se ejecuta la siguiente sentencia de SQL. Si se produce un error en un flujo XA retardado, se informa de ese error como parte de una excepción SQLException que emite la sentencia de SQL actual.	
-4210	Tiempo de espera excedido al obtener un objeto de transporte de la agrupación.	57033
-4211	Tiempo de espera excedido al obtener un objeto de la agrupación.	57033
-4212	Miembro de Sysplex no disponible.	
-4213	Tiempo de espera excedido.	57033
-4214	*texto-de-getMessage* **Explicación:** la autorización no era válida. **Respuesta de usuario:** llamar a SQLException.getMessage para recuperar información específica sobre el problema.	28000
-4220	*texto-de-getMessage* **Explicación:** se produjo un error al convertir caracteres. **Respuesta de usuario:** llamar a SQLException.getMessage para recuperar información específica sobre el problema.	
-4221	*texto-de-getMessage* **Explicación:** se produjo un error al cifrar o descifrar. **Respuesta de usuario:** llamar a SQLException.getMessage para recuperar información específica sobre el problema.	
-4222	*texto-de-getMessage* **Explicación:** se produjo un error al conectarse con la fuente de datos. **Respuesta de usuario:** llamar a SQLException.getMessage para recuperar información específica sobre el problema.	
-4223	*texto-de-getMessage* **Explicación:** se produjo un error en la inicialización. **Respuesta de usuario:** llamar a SQLException.getMessage para recuperar información específica sobre el problema.	

Tabla 117. Códigos de error emitidos por IBM Data Server Driver para JDBC y SQLJ (continuación)

Código de error	Texto del mensaje y explicación	SQLSTATE
-4224	*texto-de-getMessage* **Explicación:** se produjo un error al realizar la limpieza de recursos. **Respuesta de usuario:** llamar a SQLException.getMessage para recuperar información específica sobre el problema.	
-4225	*texto-de-getMessage* **Explicación:** se produjo un error al enviar datos a un servidor o al recibirlos de un servidor. **Respuesta de usuario:** llamar a SQLException.getMessage para recuperar información específica sobre el problema.	
-4226	*texto-de-getMessage* **Explicación:** se produjo un error durante la personalización o la vinculación. **Respuesta de usuario:** llamar a SQLException.getMessage para recuperar información específica sobre el problema.	
-4227	*texto-de-getMessage* **Explicación:** se produjo un error en el restablecimiento. **Respuesta de usuario:** llamar a SQLException.getMessage para recuperar información específica sobre el problema.	
-4228	*texto-de-getMessage* **Explicación:** se produjo un error que no se adecua en otra categoría. **Respuesta de usuario:** llamar a SQLException.getMessage para recuperar información específica sobre el problema.	
-4229	*texto-de-getMessage* **Explicación:** se produjo un error durante la ejecución de un proceso por lotes. **Respuesta de usuario:** llamar a SQLException.getMessage para recuperar información específica sobre el problema.	
-4231	Se ha producido un error al convertir la columna *número-columna* de tipo *tipo-datos-sql* con valor *valor* a un valor de tipo java.math.BigDecimal.	
-4450	No se soporta la característica: *nombre-característica*.	0A504
-4460	*texto-de-getMessage* **Explicación:** el valor especificado no es una opción válida. **Respuesta de usuario:** llamar a SQLException.getMessage para recuperar información específica sobre el problema.	
-4461	*texto-de-getMessage* **Explicación:** el valor especificado no es válido o se encuentra fuera del intervalo esperado. **Respuesta de usuario:** llamar a SQLException.getMessage para recuperar información específica sobre el problema.	42815

Tabla 117. Códigos de error emitidos por IBM Data Server Driver para JDBC y SQLJ (continuación)

Código de error	Texto del mensaje y explicación	SQLSTATE
-4462	*texto-de-getMessage* **Explicación:** falta un valor necesario. **Respuesta de usuario:** llamar a SQLException.getMessage para recuperar información específica sobre el problema.	
-4463	*texto-de-getMessage* **Explicación:** el valor especificado tiene un error de sintaxis. **Respuesta de usuario:** llamar a SQLException.getMessage para recuperar información específica sobre el problema.	42601
-4470	*texto-de-getMessage* **Explicación:** la operación solicitada no se puede realizar debido a que el recurso de destino está cerrado. **Respuesta de usuario:** llamar a SQLException.getMessage para recuperar información específica sobre el problema.	
-4471	*texto-de-getMessage* **Explicación:** la operación solicitada no se puede realizar debido a que se está utilizando el recurso de destino. **Respuesta de usuario:** llamar a SQLException.getMessage para recuperar información específica sobre el problema.	
-4472	*texto-de-getMessage* **Explicación:** la operación solicitada no se puede realizar debido a que el recurso de destino no está disponible. **Respuesta de usuario:** llamar a SQLException.getMessage para recuperar información específica sobre el problema.	
-4473	*texto-de-getMessage* **Explicación:** la operación solicitada no se puede realizar debido a que el recurso de destino ya no está disponible. **Respuesta de usuario:** llamar a SQLException.getMessage para recuperar información específica sobre el problema.	
-4474	*texto-de-getMessage* **Explicación:** la operación solicitada no se puede realizar debido a que el recurso de destino no se puede cambiar. **Respuesta de usuario:** llamar a SQLException.getMessage para recuperar información específica sobre el problema.	
-4475	*texto-de-getMessage* **Explicación:** la operación solicitada no se puede realizar debido a que el acceso al recurso de destino está restringido. **Respuesta de usuario:** llamar a SQLException.getMessage para recuperar información específica sobre el problema.	

Tabla 117. Códigos de error emitidos por IBM Data Server Driver para JDBC y SQLJ (continuación)

Código de error	Texto del mensaje y explicación	SQLSTATE
-4476	*texto-de-getMessage* **Explicación:** la operación solicitada no se puede realizar debido a que no se permite la operación en el recurso de destino. **Respuesta de usuario:** llamar a SQLException.getMessage para recuperar información específica sobre el problema.	
-4496	Se ha emitido una sentencia OPEN de SQL para un cursor retenido en una conexión XA. El controlador JDBC no permite abrir un cursor retenido en el servidor de bases de datos para una conexión XA.	
-4497	La aplicación debe emitir una retrotracción. Ya se ha retrotraído la unidad de trabajo en el servidor DB2, pero es posible que otros gestores de recursos implicados en la unidad de trabajo no hayan retrotraído sus cambios. Para asegurar la integridad de la aplicación, todas las peticiones de SQL se rechazan hasta que la aplicación emite una retrotracción.	
-4498	Ha fallado una conexión, pero se ha vuelto a establecer. Se han repetido en caso necesario los valores del registro especial. Nombre de sistema principal o dirección IP de la conexión: *nombre-sistema-principal*. Nombre de servicio o número de puerto de la conexión: *nombre-servicio*. Código de razón: *código-razón*. Código de anomalía: *código-anomalía*. Código de error: *código-error*. **Explicación:** La conexión se ha restablecido. En algunos casos, la conexión de red o el transporte al servidor no se establecen hasta el siguiente uso. Después de restablecerse la conexión, todos los recursos de la sesión están establecidos en sus valores por omisión iniciales. La aplicación se retrotrae a su punto de confirmación anterior. El código de razón indica qué valores de registro especial se han aplicado a la nueva conexión. Los valores posibles para el código de razón son: **1** Todos los valores de registro especial han vuelto a adquirir los valores que tenían en el punto de anomalía. La conexión se ha restablecido dentro del grupo actual. **2** Todos los valores de registro especial han vuelto a adquirir los valores que tenían en el punto de confirmación anterior. La conexión se ha restablecido dentro del grupo actual. **3** Todos los registros especiales han vuelto a adquirir los valores que tenían en el punto de anomalía. La conexión se ha restablecido en un nuevo grupo. **4** Todos los valores de registro especial han vuelto a adquirir los valores que tenían en el punto de confirmación anterior. La conexión se ha restablecido en un nuevo grupo. *código-anomalía* indica el error que ha dado lugar a que la conexión fallara: **1** Se ha producido un error de comunicación. **2** El servidor de datos ha cerrado la conexión. **3** Se ha producido un error de SQL. **4** El cliente ha cerrado la conexión.	

Tabla 117. Códigos de error emitidos por IBM Data Server Driver para JDBC y SQLJ (continuación)

Código de error	Texto del mensaje y explicación	SQLSTATE
-4498 (continuación)	*código-error* depende del valor de *código-anomalía*: **Código de anomalía: 1 ó 2** **Código de error:** El mensaje de SocketException de Java que se ha devuelto. **Código de anomalía: 3** **Código de error:** El código de error de SQL que ha devuelto la sentencia de SQL que ha dado lugar a que la conexión fallara. **Código de anomalía: 4** **Código de error:** Uno de los valores siguientes: **1** Se ha excedido el valor commandTimeout. **2** El controlador ha recibido una petición de interrupción o cancelación. **3** Se ha excedido el valor queryTimeout. Para el redireccionamiento de cliente en relación a servidores DB2 para z/OS, no se han restablecido los valores especiales de registro que se establecieron después del último punto de confirmación. La aplicación se retrotrae a su punto de confirmación anterior. Es posible que el estado de la conexión y los recursos globales como las tablas temporales globales y los cursores retenidos abiertos no se actualicen.	
-4499	*texto-de-getMessage* **Explicación:** se ha producido un error muy grave que ha provocado la desconexión de la fuente de datos. La conexión existente ha pasado a estar inutilizable. **Respuesta de usuario:** llamar a SQLException.getMessage para recuperar información específica sobre el problema.	08001 o 58009

Tabla 117. Códigos de error emitidos por IBM Data Server Driver para JDBC y SQLJ (continuación)

Código de error	Texto del mensaje y explicación	SQLSTATE
-30108	Ha fallado una conexión, pero se ha vuelto a establecer. Se han repetido en caso necesario los valores del registro especial. Nombre de sistema principal o dirección IP de la conexión: *nombre-sistema-principal*. Nombre de servicio o número de puerto de la conexión: *nombre-servicio*. Código de razón: *código-razón*. Código de anomalía: *código-anomalía*. Código de error: *código-error*. **Explicación:** La conexión se ha restablecido. En algunos casos, la conexión de red o el transporte al servidor no se establecen hasta el siguiente uso. Después de restablecerse la conexión, todos los recursos de la sesión están establecidos en sus valores por omisión iniciales. La aplicación se retrotrae a su punto de confirmación anterior. El código de razón indica qué valores de registro especial se han aplicado a la nueva conexión. Los valores posibles para el código de razón son: **1** Todos los valores de registro especial han vuelto a adquirir los valores que tenían en el punto de anomalía. La conexión se ha restablecido dentro del grupo actual. **2** Todos los valores de registro especial han vuelto a adquirir los valores que tenían en el punto de confirmación anterior. La conexión se ha restablecido dentro del grupo actual. **3** Todos los registros especiales han vuelto a adquirir los valores que tenían en el punto de anomalía. La conexión se ha restablecido en un nuevo grupo. **4** Todos los valores de registro especial han vuelto a adquirir los valores que tenían en el punto de confirmación anterior. La conexión se ha restablecido en un nuevo grupo. *código-anomalía* indica el error que ha dado lugar a que la conexión fallara: **1** Se ha producido un error de comunicación. **2** El servidor de datos ha cerrado la conexión. **3** Se ha producido un error de SQL. **4** El cliente ha cerrado la conexión.	08506
-30108 (continuación)	*código-error* depende del valor de *código-anomalía*: **Código de anomalía: 1 ó 2** **Código de error:** El mensaje de SocketException de Java que se ha devuelto. **Código de anomalía: 3** **Código de error:** El código de error de SQL que ha devuelto la sentencia de SQL que ha dado lugar a que la conexión fallara. **Código de anomalía: 4** **Código de error:** Uno de los valores siguientes: **1** Se ha excedido el valor commandTimeout. **2** El controlador ha recibido una petición de interrupción o cancelación. **3** Se ha excedido el valor queryTimeout.	
-99999	IBM Data Server Driver para JDBC y SQLJ ha emitido un error que no tiene asignado todavía un código de error.	

Estados de SQL emitidos por IBM Data Server Driver para JDBC y SQLJ

Los estados de SQL (SQLSTATE) comprendidos dentro del rango 46600 a 466ZZ están reservados para IBM Data Server Driver para JDBC y SQLJ.

La tabla siguiente lista los estados de SQL producidos o utilizados por IBM Data Server Driver para JDBC y SQLJ.

Tabla 118. Estados de SQL devueltos por IBM Data Server Driver para JDBC y SQLJ

Clase de estado de SQL	SQLSTATE	Descripción
01*xxx*		Aviso
02*xxx*		Ningún dato
02*xxx*	02501	La posición del cursor no es válida para una operación FETCH sobre la fila actual.
02*xxx*	02506	Error tolerable
08*xxx*		Excepción de conexión
08*xxx*	08001	El peticionario de aplicaciones no puede establecer la conexión.
08*xxx*	08003	No existe una conexión
08*xxx*	08004	El servidor de aplicaciones ha rechazado el establecimiento de la conexión
08*xxx*	08506	Excepción de redireccionamiento del cliente
0A*xxx*		Función no admitida
0A*xxx*	0A502	La acción u operación no está permitida para esta instancia de base de datos
0A*xxx*	0A504	El controlador no soporta esta característica
22*xxx*		Excepción de datos
22*xxx*	22007	La representación de caracteres de un valor de fecha y hora no es válida
22*xxx*	22021	Un carácter no pertenece al juego de caracteres codificados
23*xxx*		Violación de restricción
23*xxx*	23502	Un valor que se ha insertado en una columna o actualiza una columna es nulo, pero la columna no puede contener valores nulos.
24*xxx*		Estado de cursor no válido
24*xxx*	24501	El cursor identificado no está abierto
28*xxx*		Excepción de autorización
28*xxx*	28000	Nombre de autorización no válido.
2D*xxx*		Terminación de transacción no válida
2D*xxx*	2D521	Las operaciones COMMIT o ROLLBACK de SQL no son válidas en el entorno operativo actual.
34*xxx*		Nombre de cursor no válido
34*xxx*	34000	El nombre del cursor no es válido.
3B*xxx*		Punto de salvaguarda no válido

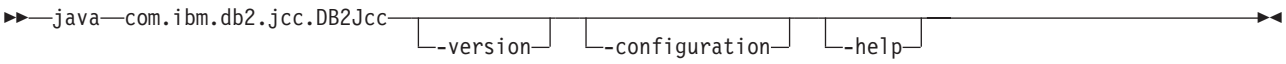
Tabla 118. Estados de SQL devueltos por IBM Data Server Driver para JDBC y SQLJ (continuación)

Clase de estado de SQL	SQLSTATE	Descripción
3B*xxx*	3B503	No está permitido utilizar una sentencia SAVEPOINT, RELEASE SAVEPOINT o ROLLBACK TO SAVEPOINT en un activador o transacción global.
40*xxx*		Retrotracción de transacción
42*xxx*		Error de sintaxis o violación de norma de acceso
42*xxx*	42601	Un carácter, símbolo o cláusula no es válido o se ha omitido
42*xxx*	42734	Se ha detectado un duplicado para nombre de parámetro, nombre de variable de SQL, nombre de cursor, nombre de condición o etiqueta.
42*xxx*	42807	La operación INSERT, UPDATE o DELETE no está permitida en este objeto
42*xxx*	42808	Una columna identificada en la operación de inserción o actualización no es actualizable
42*xxx*	42815	El tipo de datos, longitud, escala, valor o CCSID no es válido
42*xxx*	42820	Una constante numérica es demasiado larga o tiene un valor que no está dentro del rango de su tipo de datos
42*xxx*	42968	La conexión ha fallado debido a que no existe ninguna licencia de software actual.
57*xxx*		Recurso no disponible o intervención del operador
57*xxx*	57033	Se ha producido un punto muerto o un tiempo de espera excedido sin retrotracción automática
58*xxx*		Error del sistema
58*xxx*	58008	La ejecución ha fallado debido a un error de protocolo de distribución, el cual no afectará a la ejecución satisfactoria de los mandatos de DDM o sentencias de SQL subsiguientes
58*xxx*	58009	La ejecución ha fallado debido a un error de protocolo de distribución que hizo que se desasignara la conversación
58*xxx*	58012	El proceso de vinculación con el nombre de paquete y símbolo de coherencia especificados no está activo
58*xxx*	58014	El mandato de DDM no está permitido
58*xxx*	58015	El objeto de DDM no está permitido
58*xxx*	58016	El parámetro de DDM no está permitido
58*xxx*	58017	El valor del parámetro de DDM no está permitido

Búsqueda de información de versión y de entorno sobre IBM Data Server Driver para JDBC y SQLJ

Para determinar la versión de IBM Data Server Driver para JDBC y SQLJ, así como para obtener información sobre el entorno en el que se está ejecutando el controlador, ejecute el programa de utilidad DB2Jcc en la línea de mandatos de .

Sintaxis de DB2Jcc



Descripciones de las opciones de DB2Jcc

-version
Especifica que IBM Data Server Driver para JDBC y SQLJ muestra su nombre y versión.

-configuration
Especifica que IBM Data Server Driver para JDBC y SQLJ muestra su nombre y versión, así como información sobre el entorno, como la información sobre restricciones de licencia, información sobre la vía de acceso, el sistema operativo y el entorno de ejecución de Java.

-help
Especifica que el programa de utilidad DB2Jcc describe todas las opciones a las que da soporte. Si se especifica cualquier otra opción con -help, no se tiene en cuenta.

Mandatos para la preparación de programas de SQLJ

Para preparar programas SQLJ para su ejecución, utiliza mandatos que convierten el código fuente SQLJ en código fuente Java, compila el código fuente Java, crea y personaliza perfiles serializados SQLJ, y vincula paquetes DB2.

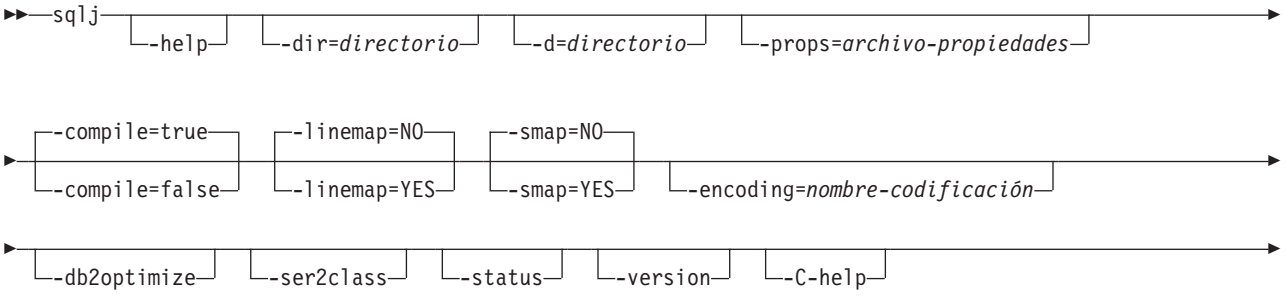
sqlj - Traductor de SQLJ

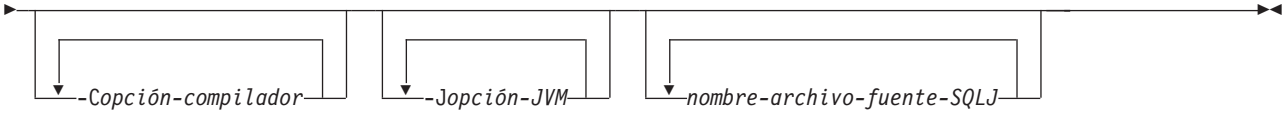
El mandato sqlj convierte un archivo fuente SQLJ en un archivo fuente Java con cero o más perfiles serializados SQLJ. Por omisión, el mandato sqlj también compila el archivo fuente Java.

Autorización

Ninguno

Sintaxis del mandato





Parámetros del mandato

-help
Especifica que el conversor de SQLJ describa cada una de las opciones a las que da soporte el traductor. Si se especifica cualquier otra opción con -help, no se tiene en cuenta.

-dir=*directorio*
Especifica el nombre del directorio donde SQLJ coloca los archivos .java producidos por el traductor y los archivos .class producidos por el compilador. El valor por omisión es el directorio donde residen los archivos fuente de SQLJ.

El traductor utiliza la estructura de directorios de los archivos fuente SQLJ cuando pone los archivos generados en directorios. Por ejemplo, suponga que desea que el traductor procese dos archivos:
- file1.sqlj, que no está en un paquete de Java
- file2.sqlj, que está en el paquete de Java sqlj.test

Además, suponga que ha especificado el parámetro -dir=/src al invocar el traductor. Éste pone el archivo fuente Java para file1.sqlj en el directorio /src y pone el archivo fuente Java para file2.sqlj en el directorio /src/sqlj/test.

-d=*directorio*
Especifica el nombre del directorio donde SQLJ coloca los archivos binarios producidos por el traductor y el compilador. Estos archivos comprenden los archivos .ser, los archivos *nombre*_SJProfileKeys.class y los archivos .class producidos por el compilador.

El valor por omisión es el directorio donde residen los archivos fuente SQLJ.

El traductor utiliza la estructura de directorios de los archivos fuente SQLJ cuando pone los archivos generados en directorios. Por ejemplo, suponga que desea que el traductor procese dos archivos:
- file1.sqlj, que no está en un paquete de Java
- file2.sqlj, que está en el paquete de Java sqlj.test

Además, suponga que ha especificado el parámetro -d=/src al invocar el traductor. Éste pone los perfiles serializados para file1.sqlj en el directorio /src y pone los perfiles serializados para file2.sqlj en el directorio /src/sqlj/test.

-compile=true|false
Especifica si el conversor de SQLJ debe compilar la fuente de Java generada en códigos de bytes.

true
El traductor compila el código fuente de Java generado. Éste es el valor por omisión.

false
El traductor no compila el código fuente de Java generado.

-linemap=no|yes
Especifica si los números de línea de las excepciones Java coinciden con los

números de línea del archivo fuente SQLJ (el archivo .sqlj), o los números de línea del archivo fuente Java generado por el conversor de SQLJ (el archivo .java).

no Los números de línea de las excepciones Java coinciden con los números de línea del archivo fuente Java. Éste es el valor por omisión.

yes
Los números de línea de las excepciones Java coinciden con los números de línea del archivo fuente SQLJ.

-smap=no|yes
Especifica si el conversor de SQLJ genera un archivo de correlación fuente (SMAP) por cada archivo fuente SQLJ. Algunas herramientas de depuración del lenguaje Java utilizan un archivo SMAP. Dicho archivo correlaciona las líneas del archivo fuente SQLJ con las líneas del archivo fuente Java generado por el conversor de SQLJ. El archivo está en el esquema de codificación Unicode UTF-8. Su formato se describe en la petición de especificación original Java (JSR) 45, que está disponible en el sitio Web:

`http://www.jcp.org`

no No genera archivos SMAP. Éste es el valor por omisión.

yes
Genera archivos SMAP. El nombre de un archivo MAP es del tipo *nombre-archivo-fuente-SQLJ*.java.smap. El conversor de SQLJ pone el archivo SMAP en el mismo directorio que el archivo fuente Java generado.

-encoding=*nombre-codificación*
Especifica la codificación del archivo fuente. Por ejemplo, JIS o EUC. Si no se especifica esta opción, se utilizará el convertidor por omisión del sistema operativo.

-db2optimize
Especifica que el conversor de SQLJ genere código que permita el almacenamiento en antememoria de contextos SQLJ en un entorno de WebSphere Application Server para las aplicaciones que se ejecutan en servidores de datos DB2.

-db2optimize hace que un contexto definido por el usuario amplíe una clase de controlador personalizada, lo que habilita el almacenamiento en antememoria de contextos y el almacenamiento en antememoria de conexiones en WebSphere Application Server.

Dado que el almacenamiento en antememoria de contextos se habilita utilizando una instancia de la clase sqlj.runtime.ref.DefaultContext de IBM Data Server Driver para JDBC y SQLJ, db2jcc.jar debe estar en la CLASSPATH cuando se traduzca y compile la aplicación Java.

No se puede utilizar el compartimiento de conexiones en WebSphere Application Server si se utiliza el almacenamiento en antememoria de contextos.

Importante: El almacenamiento en antememoria de contextos que se habilita con la opción-db2optimize puede proporcionar un rendimiento mejorado, en comparación con el almacenamiento en antememoria de conexiones y el almacenamiento en antememoria de sentencias que suministra WebSphere Application Server. No obstante, el almacenamiento en antememoria de contextos puede generar un consumo de recursos importante en el servidor de aplicaciones, y podría tener efectos secundarios inesperados si no se utiliza correctamente. Por ejemplo, si dos aplicaciones utilizan un perfil de SQLJ con

el mismo nombre, podrían sobrescribirse entre sí, ya que ambas utilizan sqlj.runtime.ref.DefaultContext. Utilice el almacenamiento en antememoria de contextos únicamente si:

- El sistema no tiene restricciones de almacenamiento.
- Las sentencias almacenadas en antememoria se reutilizan en la misma Connection.
- Todas las aplicaciones tienen nombres diferenciados para sus perfiles de SQLJ.

-ser2class
Especifica que el conversor de SQLJ convierte archivos .ser en archivos .class.

-status
Especifica que el conversor de SQLJ muestra mensajes de estado durante su ejecución.

-version
Especifica que el conversor de SQLJ muestra la versión de IBM Data Server Driver para JDBC y SQLJ. La información es de la forma:

```
IBM SQLJ xxxx.xxxx.xx
```

-C-help
Especifica que el conversor de SQLJ muestra información de ayuda para el compilador Java.

-C*opción-compilador*
Especifica una opción de compilador Java válida que empieza por un guión (-). No incluye los espacios entre -C y la opción de compilador. Si ha de especificar varias opciones de compilador, deberá anteponer -C a todas las opciones de compilador. Por ejemplo:

```
-C-g -C-verbose
```

Todas las opciones se le pasarán al compilador Java y no las utilizará el conversor de SQLJ, **a excepción** de las opciones siguientes:

-classpath
Especifica la vía de acceso a clases del usuario que van a utilizar el conversor de SQLJ y el compilador Java. Este valor altera temporalmente la variable de entorno CLASSPATH.

-sourcepath
Especifica la vía de acceso al código fuente en que el conversor de SQLJ y el compilador Java buscarán definiciones de clases o interfaces. El conversor de SQLJ busca archivos .sqlj y .java únicamente en los directorios, no en los archivos JAR o zip.

-J*opción-JVM*
Especifica una opción que se le pasará a la máquina virtual de Java (JVM) en que se ejecuta el mandato sqlj. La opción debe ser una opción de JVM válida que empieza por un guión (-). No incluye los espacios entre -J y la opción de JVM. Si ha de especificar varias opciones de JVM, deberá anteponer -J a todas las opciones de compilador. Por ejemplo:

```
-J-Xmx128m -J-Xmine2M
```

nombre-archivo-fuente-SQLJ
Especifica una lista de archivos fuente SQLJ que se deben convertir. Se trata de un parámetro obligatorio. Todos los nombres de archivos fuente SQLJ deben tener la extensión .sqlj.

Salida

Para cada archivo fuente, *nombre-programa*.sqlj, el conversor de SQLJ genera los archivos siguientes:
- El programa fuente generado

  El archivo fuente generado se denomina *nombre-programa*.java.
- Un archivo de perfiles serializados para cada clase de contexto de conexión que se utilice en una cláusula ejecutable de SQLJ

  Un nombre de perfil serializado tiene el formato siguiente:

  *nombre-programa*_SJProfile*NúmeroID*.ser
- Si el conversor de SQLJ invoca el compilador Java, los archivos de clases generados por el compilador.

Ejemplos

```
sqlj -encoding=UTF8 -C-O MyApp.sqlj
```

db2sqljcustomize - Personalizador de perfiles de SQLJ

db2sqljcustomize procesa un perfil de SQLJ, que contiene sentencias de SQL incorporado.

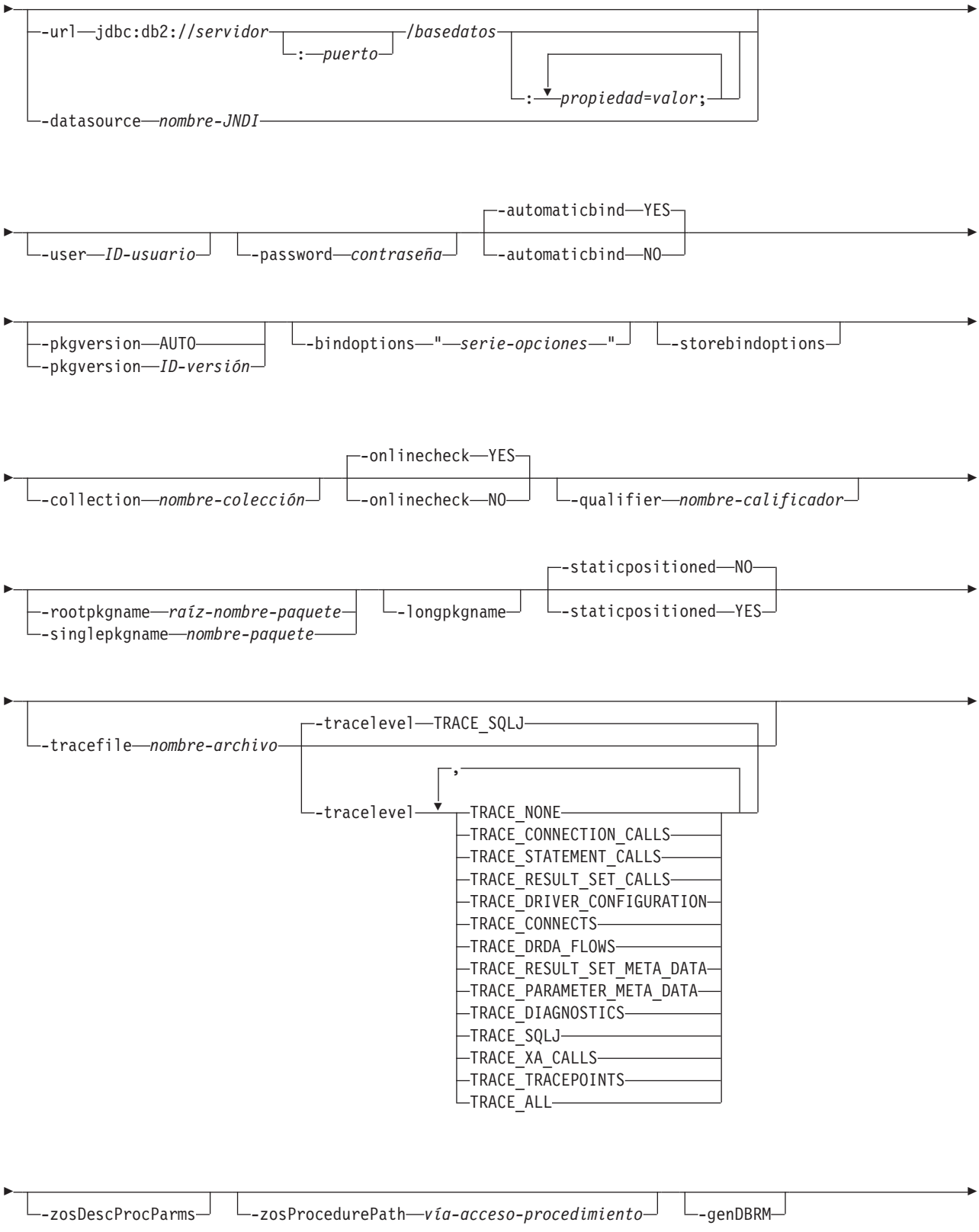
Por omisión, db2sqljcustomize produce cuatro paquetes DB2: uno para cada nivel de aislamiento. db2sqljcustomize amplia el perfil con información específica de DB2 para su utilización durante la ejecución.

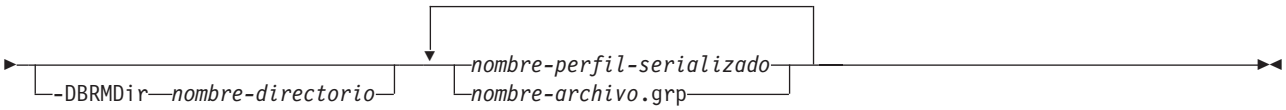
Autorización

El conjunto de privilegios del proceso tiene que incluir una de las autorizaciones posibles:
- Autorización DBADM
- Si el paquete no existe, el privilegio BINDADD y uno de los privilegios siguientes:
  - Privilegio CREATEIN
  - Autorización IMPLICIT_SCHEMA sobre la base de datos si el nombre de esquema del paquete no existe
- Si el paquete existe:
  - Privilegio ALTERIN sobre el esquema
  - Privilegio BIND sobre el paquete

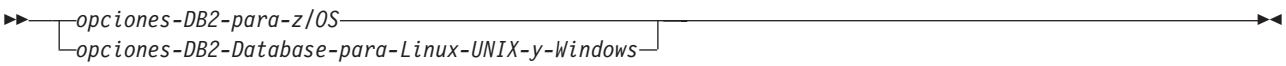
El usuario también necesita todos los privilegios necesarios para compilar las sentencias de SQL estático de la aplicación. Los privilegios otorgados a grupos no se utilizan para la comprobación de autorizaciones de sentencias estáticas.

Sintaxis del mandato

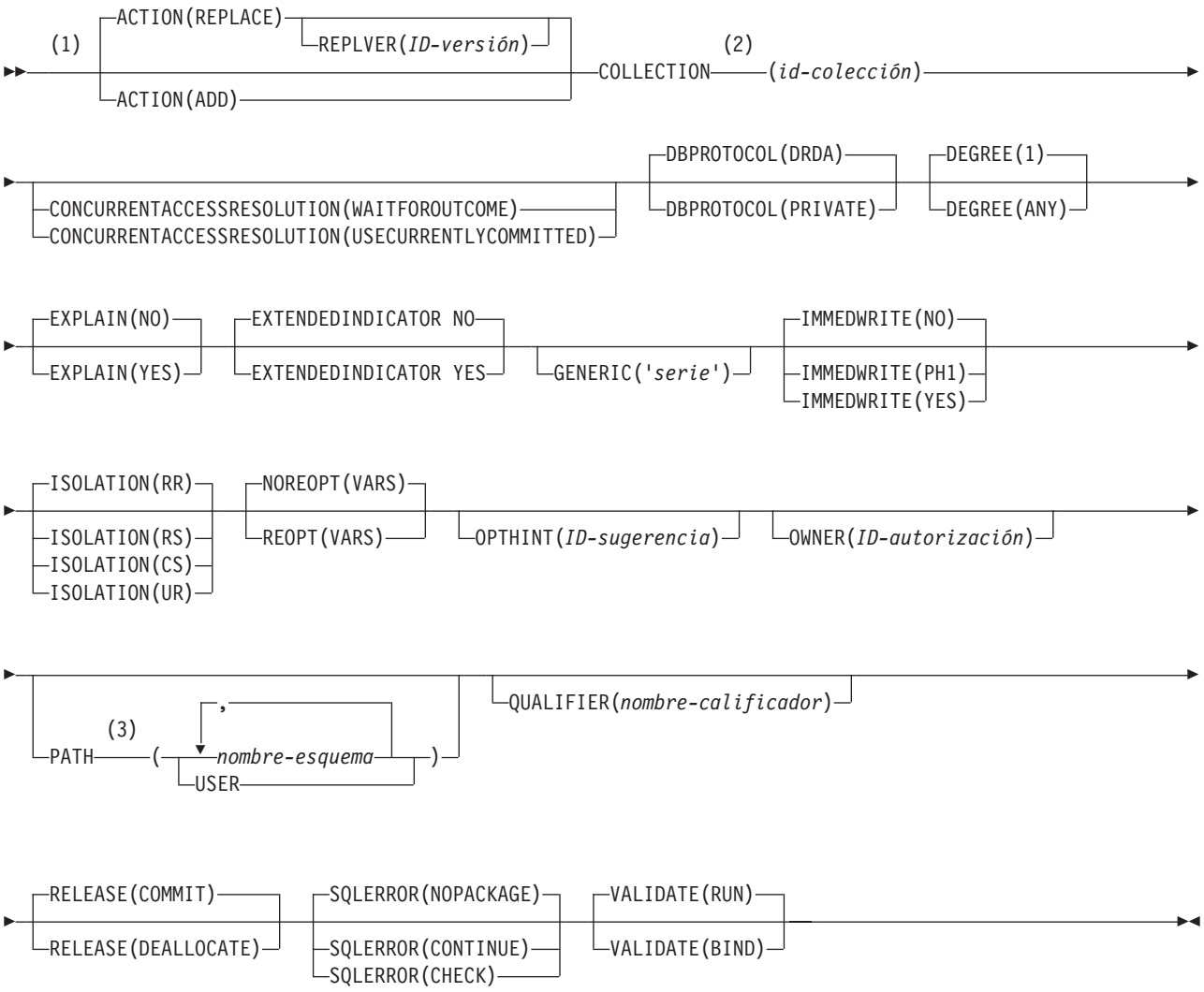

-url jdbc:db2://servidor
:puerto
/basedatos
:propiedad=valor;
-datasource nombre-JNDI
-user ID-usuario
-password contraseña
-automaticbind YES
-automaticbind NO
-pkgversion AUTO
-pkgversion ID-versión
-bindoptions "serie-opciones"
-storebindoptions
-collection nombre-colección
-onlinecheck YES
-onlinecheck NO
-qualifier nombre-calificador
-rootpkgname raíz-nombre-paquete
-singlepkgname nombre-paquete
-longpkgname
-staticpositioned NO
-staticpositioned YES
-tracefile nombre-archivo
-tracelevel TRACE_SQLJ
-tracelevel
,
TRACE_NONE
TRACE_CONNECTION_CALLS
TRACE_STATEMENT_CALLS
TRACE_RESULT_SET_CALLS
TRACE_DRIVER_CONFIGURATION
TRACE_CONNECTS
TRACE_DRDA_FLOWS
TRACE_RESULT_SET_META_DATA
TRACE_PARAMETER_META_DATA
TRACE_DIAGNOSTICS
TRACE_SQLJ
TRACE_XA_CALLS
TRACE_TRACEPOINTS
TRACE_ALL
-zosDescProcParms
-zosProcedurePath vía-acceso-procedimiento
-genDBRM



serie-opciones:



DB2 para z/OS opciones:



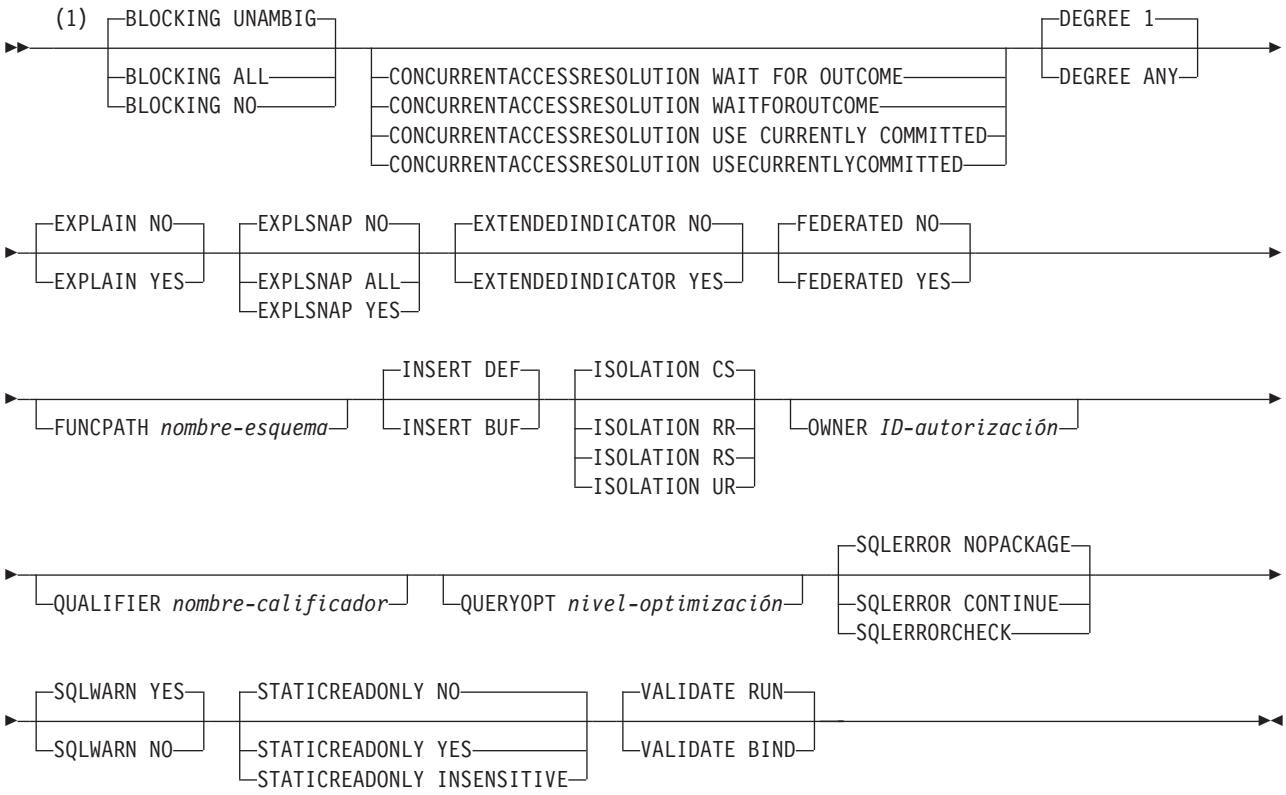
Notas:

1 Estas opciones se pueden especificar en cualquier orden.

2 COLLECTION no es una opción BIND válida para DB2 para z/OS. No obstante, se puede especificar COLLECTION como argumento de -bindoptions en db2sqljcustomize o db2sqljbind, para cambiar la colección en el momento de la vinculación, o para vincular el mismo archivo de perfil serializado en varias colecciones.

3 FUNCPATH se puede especificar como alias para PATH.

DB2 Database para Linux, UNIX y Windows opciones:



Notas:

1 Estas opciones se pueden especificar en cualquier orden.

Parámetros del mandato

-help
Especifica que el personalizador de SQLJ describe todas las opciones soportadas por el personalizador. Si se especifica cualquier otra opción con -help, no se tiene en cuenta.

-url
Especifica el URL de la fuente de datos para la que se personalizará el perfil. Se establece una conexión con la fuente de datos que este URL representa si la opción -automaticbind o -onlinecheck se especifica como YES o toma de por omisión el valor YES. Las partes variables del valor `-url` son:

servidor
Nombre de dominio o dirección IP del sistema z/OS donde reside el subsistema DB2.

puerto
Número de puerto de servidor TCP/IP asignado al subsistema DB2. El valor por omisión es 446.

-url
Especifica el URL de la fuente de datos para la que se personalizará el perfil. Se establece una conexión con la fuente de datos que este URL

representa si la opción -automaticbind o -onlinecheck se especifica como YES o toma de por omisión el valor YES. Las partes variables del valor `-url` son:

servidor
Nombre de dominio o dirección IP del sistema operativo donde reside el servidor de bases de datos.

puerto
El número de puerto del servidor TCP/IP que está asignado al servidor de bases de datos. El valor por omisión es 446.

basedatos
Nombre del servidor de bases de datos para el que se va a personalizar el perfil.

Si la conexión es con un servidor DB2 para z/OS, *basedatos* es el nombre de ubicación de DB2 que se define durante la instalación. Todos los caracteres de este valor deben ser caracteres en mayúsculas. Puede determinar el nombre de ubicación ejecutando la sentencia de SQL siguiente en el servidor:

```
SELECT CURRENT SERVER FROM SYSIBM.SYSDUMMY1;
```

Si la conexión es con un servidor DB2 Database para Linux, UNIX y Windows, *basedatos* es el nombre de la base de datos que se define durante la instalación.

Si la conexión es con un servidor IBM Cloudscape, *basedatos* es el nombre totalmente calificado del archivo donde reside la base de datos. Este nombre se debe incluir entre comillas dobles ("). Por ejemplo:

```
"c:/basedatos/testdb"
```

propiedad=valor;
Propiedad de la conexión JDBC.

propiedad=valor;
Propiedad de la conexión JDBC.

-datasource *nombre-JNDI*
Especifica el nombre lógico de un objeto DataSource que se registró con JNDI. El objeto DataSource representa la fuente de datos para la que se va personalizar el perfil. Se establece una conexión con la fuente de datos si la opción -automaticbind o -onlinecheck se especifica como YES o toma de por omisión el valor YES. La especificación de -datasource es una alternativa a especificar -url. El objeto DataSource debe representar una conexión que utilice IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4.

-user *ID-usuario*
Especifica el ID de usuario que se utilizará para conectarse con la fuente de datos para realizar comprobaciones en línea o vincular un paquete. Si especifica -url, debe especificar -user. Si especifica -datasource y el objeto DataSource que representa el *nombre-JNDI* no contiene un ID de usuario, debe especificar -user.

-password *contraseña*
Especifica la contraseña que se utilizará para conectarse con la fuente de datos para realizar comprobaciones en línea o vincular un paquete. Si especifica -url, debe especificar -password. Si especifica -datasource y el objeto DataSource que representa el *nombre-JNDI* no contiene una contraseña, debe especificar -password.

-automaticbind YES|NO
Especifica si el personalizador vincula paquetes DB2 en la fuente de datos especificada por el parámetro -url.

El valor por omisión es YES.

El número de paquetes y los niveles de aislamiento de estos paquetes están controlados por las opciones -rootpkgname y -singlepkgname.

Para que la operación de vinculación pueda funcionar se deben cumplir las condiciones siguientes:

- TCP/IP y DRDA deben estar instalados en la fuente de datos de destino.
- Deben especificarse valores válidos para -url, -username y -password.
- El valor -username debe tener autorización para vincular un paquete en la fuente de datos de destino.

-pkgversion AUTO|*ID-versión*
Especifica la versión del paquete que se va a utilizar al vincular paquetes en el servidor para el perfil serializado que se está personalizando. db2sqljcustomize almacena el ID de versión en el perfil serializado y en el paquete DB2. La verificación de la versión en tiempo de ejecución se basa en el símbolo de coherencia y no en el nombre de la versión. Para generar automáticamente un nombre de versión basado en el símbolo de coherencia, especifique -pkgversion AUTO.

El valor por omisión es que no hay versión.

-bindoptions *serie-opciones*
Especifica una lista de opciones separadas por espacios. Estas opciones tienen la misma función que las opciones de precompilación y vinculación de DB2 que tienen los mismos nombres. Si está preparando el programa para que se ejecute en un sistema DB2 para z/OS, especifique opciones de DB2 para z/OS. Si está preparando el programa para que se ejecute en un sistema DB2 Database para Linux, UNIX y Windows, especifique opciones de DB2 Database para Linux, UNIX y Windows.

Notas sobre las opciones de vinculación:

- Especifique ISOLATION sólo si también especifica la opción -singlepkgname.
- El valor de STATICREADONLY es YES para los servidores que dan soporte a STATICREADONLY, y NO para los demás servidores. Cuando especifica STATICREADONLY YES, DB2 procesa los cursores ambiguos como si fueran cursores de solo lectura. Para la resolución de problemas de errores de declaración de iterador, tendrá que especificar explícitamente STATICREADONLY NO, o declarar iteradores de forma que no sean ambiguos. Por ejemplo, si desea que un iterador se pueda actualizar de forma no ambigua, declare el iterador para implementar sqlj.runtime.ForUpdate. Si desea un iterador de solo lectura, incluya la cláusula FOR READ ONLY en las sentencias SELECT que utilicen el iterador.

Importante: especifique solamente las opciones de preparación del programa que sean adecuadas para la fuente de datos en que se está vinculando el paquete. Algunos valores explícitos y valores por omisión utilizados para IBM Data Server Driver para JDBC y SQLJ son diferentes de los valores explícitos y valores por omisión utilizados para DB2.

-storebindoptions
Especifica que los valores para los parámetros -bindoptions y -staticpositioned se almacenan en el perfil serializado. Si db2sqljbind se invoca sin los parámetros -bindoptions o -staticpositioned, los valores que se almacenan en el

perfil serializado se utilizan durante la operación de vinculación. Cuando se especifican varios perfiles serializados para una invocación de db2sqljcustomize, los valores de los parámetros se almacenan en cada perfil serializado. Los valores almacenados se visualizan en la salida del programa de utilidad db2sqljprint.

-collection *nombre-colección*
El calificador para los paquetes vinculados por `db2sqljcustomize`. `db2sqljcustomize` almacena este valor en el perfil serializado personalizado y se utiliza cuando se vinculan los paquetes asociados. Si no especifica este parámetro, `db2sqljcustomize` utiliza NULLID como ID de colección.

-onlinecheck YES|NO
Especifica si se van a ejecutar las comprobaciones en línea de tipos de datos en el programa SQLJ. La opción -url o -datasource determina la fuente datos que se va a utilizar para las comprobaciones en línea. El valor por omisión es YES si se especifica el parámetro -url o -datasource. En otro caso, el valor por omisión es NO.

-qualifier *nombre-calificador*
Especifica el calificador que se va a utilizar para objetos no calificados en el programa SQLJ durante las comprobaciones en línea. Este valor no se utiliza como calificador cuando se vinculan paquetes.

-rootpkgname|-singlepkgname
Especifica los nombres de los paquetes que están asociados al programa. Si -automaticbind es NO, se utilizan estos nombres de paquetes al ejecutar `db2sqljbind`. Los significados de los parámetros son:

-rootpkgname *raíz-nombre-paquete*
Especifica que el personalizador crea cuatro paquetes: uno para cada uno de los cuatro niveles de aislamiento de DB2. Los nombres de los cuatro paquetes son:

*raíz-nombre-paquete***1**
Para el nivel de aislamiento UR

*raíz-nombre-paquete***2**
Para el nivel de aislamiento CS

*raíz-nombre-paquete***3**
Para el nivel de aislamiento RS

*raíz-nombre-paquete***4**
Para el nivel de aislamiento RR

Si no se especifica -longpkgname, *raíz-nombre-paquete* deberá ser una serie alfanumérica de siete o menos bytes.

Si se especifica -longpkgname, *raíz-nombre-paquete* deberá ser una serie alfanumérica de 127 o menos bytes.

-singlepkgname *nombre-paquete*
Especifica que el personalizador crea un paquete con el nombre *nombre-paquete*. Si especifica esta opción, el programa sólo podrá ejecutarse en un nivel de aislamiento. El nivel de aislamiento del paquete se especifica entrando la opción ISOLATION en la serie de opciones -bindoptions.

Si no se especifica -longpkgname, *nombre-paquete* deberá ser una serie alfanumérica de ocho o menos bytes.

Si se especifica -longpkgname, *nombre-paquete* deberá ser una serie alfanumérica de 128 o menos bytes.

No se recomienda utilizar la opción -singlepkgname.

Recomendación: si la fuente de datos es DB2 para z/OS, utilice caracteres en mayúsculas para el valor *raíz-nombre-paquetes* o para el valor *nombre-paquete*. Los sistemas DB2 para z/OS que se definen con determinados valores CCSID no admiten caracteres en minúsculas en los nombres de colección o en los nombres de paquetes.

Si no especifica -rootpkgname o -singlepkgname, `db2sqljcustomize` genera cuatro nombres de paquete basados en el nombre del perfil serializado. Un nombre de perfil serializado tiene el formato siguiente:

*nombre-programa*_SJProfile*NúmeroID*.ser

Los cuatro nombres de paquete generados tienen el formato siguiente:

Bytes-de-nombre-programaNúmeroIDAislamPaquete

La Tabla 119 muestra las partes de un nombre de paquete generado y el número de bytes de cada parte.

La longitud máxima de un nombre de paquete es *lonmáx*. *lonmáx* es 8 si no se especifica -longpkgname. *lonmáx* es 128 si se especifica -longpkgname.

Tabla 119. Partes de un nombre de paquete generado por db2sqljcustomize

Parte del nombre del paquete	Número de bytes	Valor
Bytes-del-nombre-programa	*m*=min(Length(*nombre-programa*), *lonmáx*–1–Length(*NúmeroID*))	Los primeros *m* bytes del *nombre-programa*, en mayúsculas
NúmeroID	Length(*NúmeroID*)	*NúmeroID*
Aislamientopqt	1	1, 2, 3 ó 4. Este valor representa el nivel de aislamiento de la transacción para el paquete. Consulte la Tabla 120.

La Tabla 120 muestra los valores de la porción *Aislamientopqt* de un nombre de paquete generado por db2sqljcustomize.

Tabla 120. Valores de Aislamientopqt y niveles de aislamiento asociados

Valor de *Númeropqt*	Nivel de aislamiento del paquete
1	Lectura no confirmada (UR)
2	Estabilidad del cursor (CS)
3	Estabilidad de lectura (RS)
4	Lectura repetible (RR)

Ejemplo: supongamos que el nombre de un perfil sea ThisIsMyProg_SJProfile111.ser. No se especifica la opción -longpkgname de db2sqljcustomize. Por consiguiente, *Bytes-del-nombre-programa* corresponde a los cuatro primeros bytes de ThisIsMyProg, convertidos a mayúsculas o THIS. *NúmeroID* es 111. Los nombres de los cuatro paquetes son:

```
THIS1111
THIS1112
THIS1113
THIS1114
```

Ejemplo: supongamos que el nombre de un perfil sea ThisIsMyProg_SJProfile111.ser. Se especifica la opción -longpkgname de db2sqljcustomize. Por consiguiente, *Bytes-del-nombre-programa* corresponde a ThisIsMyProg, convertidos a mayúsculas, o THISISMYPROG. *NúmeroID* es 111. Los nombres de los cuatro paquetes son:

```
THISISMYPROG1111
THISISMYPROG1112
THISISMYPROG1113
THISISMYPROG1114
```

Ejemplo: supongamos que el nombre de un perfil sea A_SJProfile0.ser. *Bytes-del-nombre-programa* es A. *NúmeroID* es 0. Por consiguiente, los nombres de los cuatro paquetes son:

```
A01
A02
A03
A04
```

No se recomienda dejar que `db2sqljcustomize` genere los nombres de paquetes. Si algún nombre de paquete generado coincide con el nombre de un paquete existente, `db2sqljcustomize` sobregrabará el paquete existente. Si desea asegurarse de la exclusividad de los nombres de paquetes, especifique -rootpkgname.

-longpkgname
Especifica que los nombres de los paquetes DB2 creados por db2sqljcustomize pueden tener hasta 128 bytes. Utilice esta opción sólo si vincula paquetes en un servidor que soporta nombres de paquetes largos. Si especifica -singlepkgname o -rootpkgname, también deberá especificar -longpkgname en las condiciones siguientes:
- El argumento de -singlepkgname tiene más de ocho bytes.
- El argumento de -rootpkgname tiene más de siete bytes

-staticpositioned NO|YES
Para los iteradores declarados en el mismo archivo fuente que las sentencias UPDATE de posición que utilizan los iteradores, especifica si las sentencias UPDATE de posición se ejecutarán como sentencias vinculadas estáticamente. El valor por omisión es NO. NO significa que las sentencias UPDATE posicionadas se ejecutan como sentencias vinculadas dinámicamente.

-zosDescProcParms
Especifica que db2sqljcustomize emita consultas para el catálogo de DB2 en la fuente de datos de destino a fin de determinar los tipos de datos de parámetros de SQL que se corresponden con las variables del lenguaje principal en sentencias CALL.

-zosDescProcParms se aplica únicamente a los programas que se ejecutan en servidores de datos DB2 para z/OS.

Si se ha especificado -zosDescProcParms y el ID de autorización en el que se ejecuta db2sqljcustomize no tiene acceso de lectura a la tabla de catálogos SYSIBM.SYSROUTINES, db2sqljcustomize devuelve un error y utiliza los tipos de datos de variable del lenguaje principal en las sentencias CALL para determinar los tipos de datos de SQL.

Al especificar -zosDescProcParms se puede obtener un uso más eficiente del almacenamiento en el tiempo de ejecución. Si hay disponible información de los tipos de datos de SQL, SQLJ tiene información sobre la longitud y la precisión de los parámetros INOUT y OUT, de modo que sólo asigna la cantidad de memoria necesaria para dichos parámetros. La disponibilidad de información de los tipos de datos de SQL puede tener una repercusión muy

grande sobre el uso del almacenamiento para los parámetros INOUT de carácter, los parámetros LOB OUT y los parámetros OUT decimales.

Cuando se especifica -zosDescProcParms, el servidor de datos DB2 utiliza el valor especificado o valor por omisión de -zosProcedurePath para resolver los nombres no calificados de los procedimientos almacenados para los que se solicita información de tipos de datos de SQL.

Si no se especifica -zosDescProcParms, db2sqljcustomize utiliza los tipos de datos de variable del lenguaje principal en las sentencias CALL para determinar los tipos de datos de SQL. Si db2sqljcustomize determina el tipo de datos de SQL incorrecto, puede producirse un error en el tiempo de ejecución. Por ejemplo, si el tipo de variable del lenguaje principal de Java es String, y el tipo de parámetro de procedimiento almacenado correspondiente es VARCHAR FOR BIT DATA, podría producirse un error de tiempo de ejecución de SQL, como el -4220.

-zosProcedurePath *vía-acceso-procedimiento*
Especifica una lista de nombres de esquema que DB2 para z/OS utiliza para resolver nombres de procedimientos almacenados no calificados durante la comprobación en línea de un programa SQLJ.

-zosProcedurePath sólo se aplica a los programas que se deben ejecutar sobre servidores de bases de datos DB2 para z/OS.

La lista es un valor de serie que es una lista separada por comas de nombres de esquema incluida entre comillas dobles. El servidor de bases de datos DB2 inserta esa lista en la vía de acceso de SQL para la resolución de nombres de procedimientos almacenados no calificados. La vía de acceso de SQL es:

```
SYSIBM, SYSFUN, SYSPROC, vía-acceso-procedimiento, nombre-calificador, ID-usuario
```

nombre-calificador es el valor del parámetro -qualifier e *ID-usuario* es el valor del parámetro -user.

El servidor de bases de datos DB2 prueba, de izquierda a derecha, los nombres de esquema contenidos en la vía de acceso de SQL hasta que encuentra una coincidencia con el nombre de un procedimiento almacenado que existe en ese servidor de bases de datos. Si el servidor de bases de datos DB2 encuentra una coincidencia, obtiene la información sobre los parámetros de ese procedimiento almacenado a partir del catálogo DB2. Si el servidor de bases de datos DB2 no encuentra una coincidencia, SQLJ establece los datos del parámetro sin ninguna información del catálogo DB2.

Si -zosProcedurePath no está especificado, el servidor de bases de datos DB2 utiliza esta vía de acceso de SQL:

```
SYSIBM, SYSFUN, SYSPROC, nombre-calificador, ID-usuario
```

Si no se especifica el parámetro -qualifier, la vía de acceso de SQL no incluirá *nombre-calificador*.

-genDBRM
Especifica que db2sqljcustomize genere módulos de petición de base de datos (DBRM). Estos DBRM se pueden utilizar para crear paquetes de DB2 para z/OS.

-genDBRM sólo se aplica a los programas que se deben ejecutar sobre servidores de bases de datos DB2 para z/OS.

Si están especificados -genDBRM y -automaticbind NO, db2sqljcustomize crea los DBRM, pero no los vincula para formar paquetes DB2. Si están

especificados -genDBRM y -automaticbind YES, db2sqljcustomize crea los DBRM y los vincula para formar paquetes DB2.

Se crea un DBRM para cada nivel de aislamiento de DB2. El convenio de denominación para los archivos DBRM generados es el mismo que para los paquetes. Por ejemplo, si se especifica -rootpkgname SQLJSA0 y también se especifica -genDBRM, los nombres de los cuatro archivos DBRM serán los siguientes:
- SQLJSA01
- SQLJSA02
- SQLJSA03
- SQLJSA04

-DBRMDir *nombre-directorio*
Cuando se especifica -genDBRM, -DBRMDir especifica el directorio local en el que db2sqljcustomize coloca los archivos DBRM generados. El valor por omisión es el directorio activo.

-DBRMdir sólo se aplica a los programas que se deben ejecutar sobre servidores de bases de datos DB2 para z/OS.

-tracefile *nombre-archivo*
Habilita el rastreo e identifica el archivo de salida para la información de rastreo. Esta opción debe especificarse solamente bajo la dirección del soporte de software de IBM.

-tracelevel
Si se especifica -tracefile, indica lo que se desea rastrear durante la ejecución de `db2sqljcustomize`. El valor por omisión es TRACE_SQLJ. Esta opción debe especificarse solamente bajo la dirección del soporte de software de IBM.

nombre-perfil-serializado|*nombre-archivo*`.grp`
Especifica los nombres de uno o varios perfiles serializados que se van a personalizar. El perfil serializado especificado debe residir en un directorio contenido en la variable de entorno CLASSPATH.

Un nombre de perfil serializado tiene el formato siguiente:

nombre-programa`_SJProfile`*NúmeroID*`.ser`

Puede especificar el nombre del perfil serializado con o sin la extensión .ser.

nombre-programa es el nombre del programa fuente de SQLJ, sin la extensión .sqlj. *n* es un entero entre 0 y *m-1*, donde *m* es el número de perfiles serializados generados por el conversor de SQLJ a partir del programa fuente de SQLJ.

Puede especificar nombres de perfiles serializados de una de las maneras siguientes:
- Liste los nombres en el mandato db2sqljcustomize. Si desea especificar varios nombres de perfiles serializados deben estar separados por espacios.
- Especifique los nombres de perfiles serializados, uno en cada línea, en un archivo con el nombre *nombre-archivo*.grp y especifique *nombre-archivo*.grp en el mandato db2sqljcustomize.

Si especifica más de un nombre de perfil serializado, y especifica o utiliza el valor por omisión -automaticbind YES, db2sqljcustomize vincula un solo paquete DB2 a partir de los perfiles. Cuando utiliza db2sqljcustomize para crear un paquete DB2 individual a partir de varios perfiles serializados, debe también especificar la opción -rootpkgname o -singlepkgname.

Si especifica más de un nombre de perfil serializado, y especifica -automaticbind NO, si desea vincular los perfiles serializados para formar un paquete DB2 individual al ejecutar db2sqljbind, debe especificar la misma lista de nombres de perfiles serializados, en el mismo orden, en db2sqljcustomize y db2sqljbind.

Si especifica uno o varios archivos *nombre-archivo*.grp y especifica -automaticbind NO, al ejecutar db2sqljbind, debe indicar la misma lista de archivos y en el mismo orden en el que se personalizaron los archivos.

No se puede ejecutar db2sqljcustomize en archivos individuales y agrupar estos archivos al ejecutar db2sqljbind.

Salida

Cuando se ejecuta `db2sqljcustomize`, crea un perfil serializado personalizado. También crea paquetes DB2, si el valor de automaticbind es YES.

Ejemplos

```
db2sqljcustomize -user richler -password mordecai
  -url jdbc:db2:/server:50000/sample -collection duddy
  -bindoptions "EXPLAIN YES" pgmname_SJProfile0.ser
```

Notas de uso

Siempre se recomiendan las comprobaciones en línea: Es muy recomendable utilizar las comprobaciones en línea al personalizar los perfiles serializados. La comprobación en línea determina información sobre los tipos de datos y las longitudes de variables de lenguaje principal de DB2, y es especialmente importante para los elementos siguientes:

- Predicados con variables del lenguaje principal java.lang.String y columnas CHAR

  A diferencia de las variables de caracteres en otros lenguajes principales, las variables de serie Java del lenguaje principal no se declaran con un atributo de longitud. Para optimizar debidamente una consulta que contiene variables de lenguaje principal, DB2 necesita conocer la longitud de esas variables. Por ejemplo, suponga que una consulta tiene un predicado en el que una variable del lenguaje principal String se compara con una columna CHAR y se define un índice en dicha columna. Si DB2 no puede determinar la longitud de la variable de lenguaje principal, podría realizar una exploración de espacio de tablas en lugar de una exploración de índice. Las comprobaciones en línea evitan este problema al proporcionar las longitudes de las columnas de caracteres correspondientes.
- Predicados con variables del lenguaje principal java.lang.String y columnas GRAPHIC

  Cuando no se utiliza la comprobación en línea, DB2 puede emitir un error de vinculación (SQLCODE -134) cuando encuentra un predicado en el que una variable de lenguaje principal de tipo String se compara con una columna de tipo GRAPHIC.
- Nombres de columna en la tabla de resultados de una sentencia SQLJ SELECT en un servidor remoto:

  Sin comprobaciones en línea, el controlador no puede determinar los nombres de columna para la tabla de resultados de una sentencia SELECT remota.

Personalización de varios perfiles serializados juntos: Se pueden personalizar juntos varios perfiles serializados para crear un paquete DB2 individual. En este

caso, y si especifica -staticpostioned YES, cualquier sentencia UPDATE o DELETE posicionada que haga referencia a un cursor declarado *anteriormente en el paquete* se ejecuta estáticamente, aunque la sentencia UPDATE o DELETE se encuentre en un archivo fuente distinto al de la declaración de cursor. Si desea obtener el comportamiento de-staticpositioned YES cuando el programa consiste en varios archivos fuente, debe ordenar los perfiles en el mandato db2sqljcustomize para que las declaraciones de cursor se coloquen por delante de sentencias UPDATE o DELETE posicionadas en el paquete. Para hacerlo, liste los perfiles que contengan sentencias SELECT que asignen tablas de resultados a iteradores *antes* de los perfiles que contengan las sentencias UPDATE o DELETE posicionadas que hagan referencia a dichos iteradores.

Utilización de un perfil serializado personalizado en una fuente de datos personalizada en otra fuente de datos: Puede ejecutar `db2sqljcustomize` para crear un perfil serializado personalizado para un programa de SQLJ en una fuente de datos y, a continuación, utilizar ese perfil en otra fuente de datos. Para ello debe ejecutar `db2sqljbind` varias veces en los perfiles serializados personalizados que haya creado al ejecutar `db2sqljcustomize` una vez. Cuando ejecuta los programas en estas fuentes de datos, los objetos DB2 a los que acceden los programas deben ser idénticos en cada fuente de datos. Por ejemplo, las tablas de todos los orígenes de datos deben tener los mismos esquemas de codificación y las mismas columnas con los mismos tipos de datos.

Utilización del parámetro -collection: `db2sqljcustomize` almacena el nombre de colección de DB2 en cada perfil serializado personalizado que produce. Cuando se ejecuta un programa de SQLJ, el controlador utiliza el nombre de colección almacenado en el perfil serializado personalizado para buscar paquetes que pueda ejecutar. El nombre que se almacena en el perfil serializado personalizado viene determinado por el valor del parámetro -collection. Sólo se puede almacenar un ID de colección en el perfil serializado. No obstante, puede vincular el mismo perfil serializado en varias colecciones de paquetes especificando la opción COLLECTION en el parámetro -bindoptions. Para ejecutar un paquete que se encuentre en una colección distinta a la colección especificada en el perfil serializado, incluya una sentencia SET CURRENT PACKAGESET en el programa.

Utilización del parámetro VERSION: Utilice el parámetro VERSION para vincular dos o más versiones de un paquete para el mismo programa de SQLJ en la misma colección. Podría hacerlo si ha cambiado un programa fuente de SQLJ y desea ejecutar la versión antigua y nueva del programa.

Para mantener dos versiones de un paquete, siga estos pasos:

1. Cambie el código en el programa fuente.
2. Convierta el programa fuente para crear un nuevo perfil serializado. Asegúrese de no sobregrabar el perfil serializado original.
3. Ejecute `db2sqljcustomize` para personalizar el perfil serializado y cree paquetes DB2 con los mismos nombres de paquete y en la misma colección que los paquetes originales. Para ello, utilice los mismos valores para -rootpkgname y -collection cuando vincule los paquetes nuevos que haya utilizado al crear los paquetes originales. Especifique la opción VERSION en el parámetro -bindoptions para colocar un ID de versión en el nuevo perfil serializado personalizado y en los nuevos paquetes.

   Es esencial especificar la opción VERSION al efectuar este paso. En caso contrario, sobregrabará los paquetes originales.

Cuando ejecuta la versión antigua del programa, DB2 carga las versiones antiguas de los paquetes. Cuando ejecuta la versión nueva del programa, DB2 carga las versiones nuevas de los paquetes.

Vinculación de paquetes y planes en DB2 para z/OS: Puede utilizar el parámetro -genDBRM de db2sqljcustomize para crear DBRM en el sistema local. Luego puede transferir dichos DBRM a un sistema DB2 para z/OS y vincularlos en paquetes en dicho sistema. Si piensa utilizar esta técnica, deberá transferir los archivos DBRM al sistema z/OS como archivos **binarios**, a un conjunto de datos particionados con el formato de registro FB y la longitud de registro 80. Cuando vincule paquetes, deberá especificar los siguientes valores de opciones de vinculación:

ENCODING(EBCDIC)
IBM Data Server Driver para JDBC y SQLJ en DB2 para z/OS requiere codificación EBCDIC para los paquetes.

DYNAMICRULES(BIND)
Esta opción garantiza reglas de autorización coherentes cuando SQLJ utilice SQL dinámico. SQLJ utiliza SQL dinámico para operaciones UPDATE o DELETE de posición que impliquen varios programas SQLJ.

Comportamiento de la opción de vinculación EXTENDEDINDICATOR: Si la opción de vinculación EXTENDEDINDICATOR no se especifica en la serie de opciones -bindoptions y el servidor de datos de destino soporta los indicadores ampliados, las operaciones de vinculación utilizan EXTENDEDINDICATOR(YES). Si EXTENDEDINDICATOR(NO) se ha especificado explícitamente y la aplicación contiene la sintaxis de indicador ampliado, el comportamiento puede ser inesperado ya que IBM Data Server Driver para JDBC y SQLJ trata los indicadores ampliados como valores NULL.

db2sqljbind - Vinculador de perfiles de SQLJ

db2sqljbind vincula paquetes DB2 para un perfil serializado que previamente se ha personalizado mediante el mandato db2sqljcustomize.

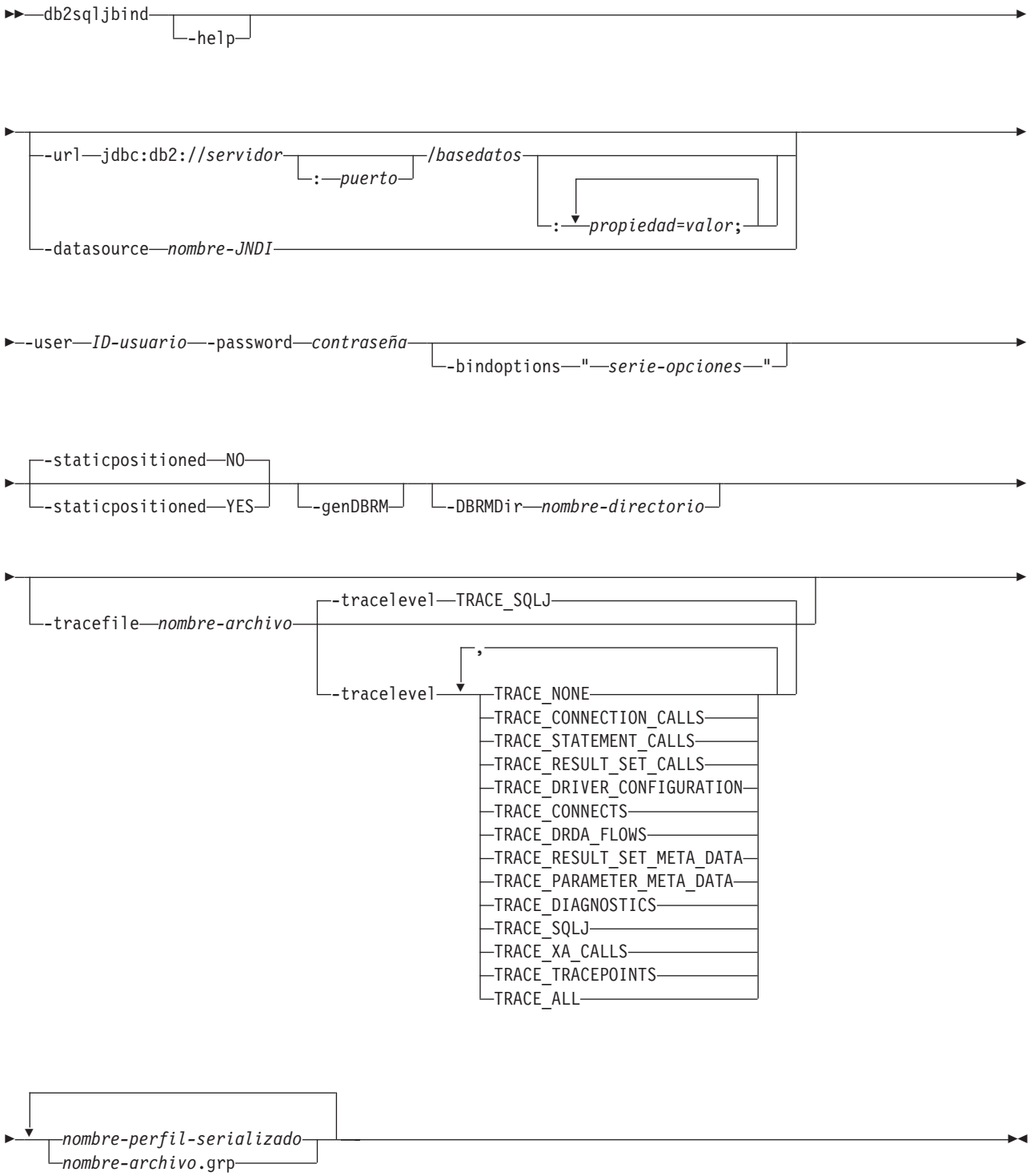
Las aplicaciones que se ejecutan con IBM Data Server Driver para JDBC y SQLJ requieren paquetes, pero no planes. Si se especifica YES en la opción -automaticbind de db2sqljcustomize, db2sqljcustomize vincula los paquetes en la fuente de datos que especifique con el parámetro -url. Sin embargo, si -automaticbind es NO, si una vinculación falla al ejecutarse db2sqljcustomize, o si desea crear paquetes idénticos en varias ubicaciones para el mismo perfil serializado, puede utilizar el mandato db2sqljbind para vincular paquetes.

Autorización

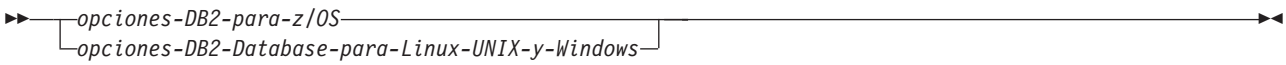
El conjunto de privilegios del proceso tiene que incluir una de las autorizaciones posibles:
- Autorización DBADM
- Si el paquete no existe, el privilegio BINDADD y uno de los privilegios siguientes:
  - Privilegio CREATEIN
  - Autorización IMPLICIT_SCHEMA sobre la base de datos si el nombre de esquema del paquete no existe
- Si el paquete existe:
  - Privilegio ALTERIN sobre el esquema
  - Privilegio BIND sobre el paquete

El usuario también necesita todos los privilegios necesarios para compilar las sentencias de SQL estático de la aplicación. Los privilegios otorgados a grupos no se utilizan para la comprobación de autorizaciones de sentencias estáticas.

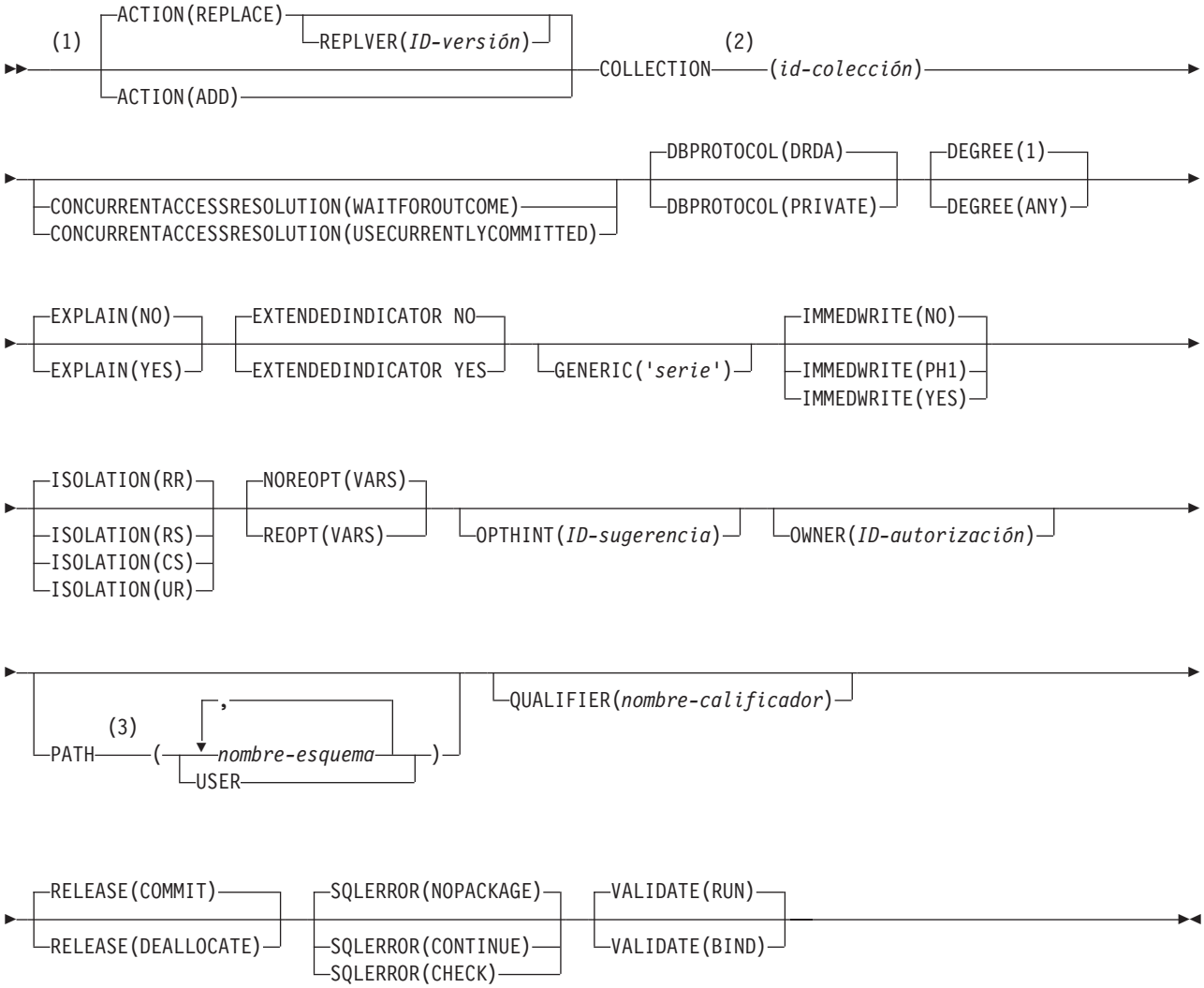
Sintaxis del mandato



serie-opciones:



DB2 para z/OS opciones:



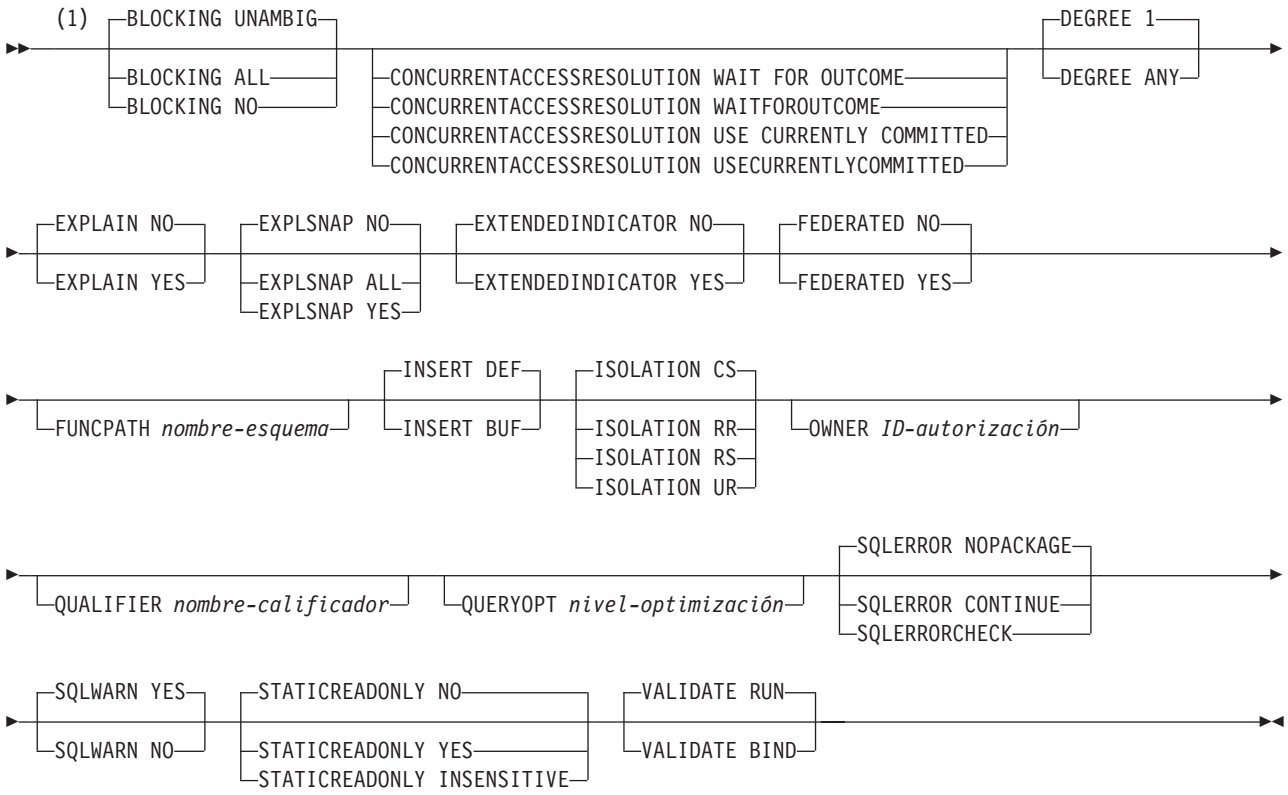
Notas:

1 Estas opciones se pueden especificar en cualquier orden.

2 COLLECTION no es una opción BIND válida para DB2 para z/OS. No obstante, se puede especificar COLLECTION como argumento de -bindoptions en db2sqljcustomize o db2sqljbind, para cambiar la colección en el momento de la vinculación, o para vincular el mismo archivo de perfil serializado en varias colecciones.

3 FUNCPATH se puede especificar como alias para PATH.

DB2 Database para Linux, UNIX y Windows opciones



Notas:

1 Estas opciones se pueden especificar en cualquier orden.

Parámetros del mandato

-help
Especifica que db2sqljbind describe todas las opciones a las que da soporte. Si se especifica cualquier otra opción con -help, no se tiene en cuenta.

-url
Especifica el URL de la fuente de datos para la que se personalizará el perfil. Se establece una conexión con la fuente de datos que este URL representa si la opción -automaticbind o -onlinecheck se especifica como YES o toma de por omisión el valor YES. Las partes variables del valor `-url` son:

servidor
Nombre de dominio o dirección IP del sistema operativo donde reside el servidor de bases de datos.

puerto
El número de puerto del servidor TCP/IP que está asignado al servidor de bases de datos. El valor por omisión es 446.

basedatos
Nombre del servidor de bases de datos para el que se va a personalizar el perfil.

Si la conexión es con un servidor DB2 para z/OS, *basedatos* es el nombre de ubicación de DB2 que se define durante la instalación. Todos los caracteres de este valor deben ser caracteres en mayúsculas. Puede determinar el nombre de ubicación ejecutando la sentencia de SQL siguiente en el servidor:

```
SELECT CURRENT SERVER FROM SYSIBM.SYSDUMMY1;
```

Si la conexión es con un servidor DB2 Database para Linux, UNIX y Windows, *basedatos* es el nombre de la base de datos que se define durante la instalación.

Si la conexión es con un servidor IBM Cloudscape, *basedatos* es el nombre totalmente calificado del archivo donde reside la base de datos. Este nombre se debe incluir entre comillas dobles ("). Por ejemplo:

```
"c:/basedatos/testdb"
```

propiedad=valor;
Propiedad de la conexión JDBC.

-datasource *nombre-JNDI*
Especifica el nombre lógico de un objeto DataSource que se registró con JNDI. El objeto DataSource representa la fuente de datos para la que se va personalizar el perfil. Se establece una conexión con la fuente de datos si la opción -automaticbind o -onlinecheck se especifica como YES o toma de por omisión el valor YES. La especificación de -datasource es una alternativa a especificar -url. El objeto DataSource debe representar una conexión que utilice IBM Data Server Driver para JDBC y SQLJ con conectividad de tipo 4.

-user *ID-usuario*
Especifica el ID de usuario que se utilizará para conectarse con la fuente de datos para vincular el paquete.

-password *contraseña*
Especifica la contraseña que se utilizará para conectarse con la fuente de datos para vincular el paquete.

-bindoptions *serie-opciones*
Especifica una lista de opciones separadas por espacios. Estas opciones tienen la misma función que las opciones de precompilación y vinculación de DB2 que tienen los mismos nombres. Si está preparando el programa para que se ejecute en un sistema DB2 para z/OS, especifique opciones de DB2 para z/OS. Si está preparando el programa para que se ejecute en un sistema DB2 Database para Linux, UNIX y Windows, especifique opciones de DB2 Database para Linux, UNIX y Windows.

Notas sobre las opciones de vinculación:

- Especifique VERSION solamente si se cumplen las condiciones siguientes son:
  - Si está vinculando un paquete en un sistema DB2 Database para Linux, UNIX y Windows, el sistema es de la versión 8 o posterior.
  - Ha vuelto a ejecutar el conversor en un programa antes de vincular el paquete asociado con un valor VERSION nuevo.
- El valor de STATICREADONLY es YES para los servidores que dan soporte a STATICREADONLY, y NO para los demás servidores. Cuando especifica STATICREADONLY YES, DB2 procesa los cursores ambiguos como si fueran cursores de solo lectura. Para la resolución de problemas de errores de declaración de iterador, tendrá que especificar explícitamente STATICREADONLY NO, o declarar iteradores de forma que no sean ambiguos. Por ejemplo, si desea que un iterador se pueda actualizar de forma no ambigua, declare el iterador para implementar sqlj.runtime.ForUpdate. Si desea un iterador de solo lectura, incluya la cláusula FOR READ ONLY en las sentencias SELECT que utilicen el iterador.

Importante: especifique solamente las opciones de preparación del programa que sean adecuadas para la fuente de datos en que se está vinculando el paquete. Algunos valores explícitos y valores por omisión utilizados para IBM Data Server Driver para JDBC y SQLJ son diferentes de los valores explícitos y valores por omisión utilizados para DB2.

-staticpositioned NO|YES
Para los iteradores declarados en el mismo archivo fuente que las sentencias UPDATE de posición que utilizan los iteradores, especifica si las sentencias UPDATE de posición se ejecutarán como sentencias vinculadas estáticamente. El valor por omisión es NO. NO significa que las sentencias UPDATE posicionadas se ejecutan como sentencias vinculadas dinámicamente. Este valor debe ser igual que el valor -staticpositioned de la invocación db2sqljcustomize anterior del perfil serializado.

-genDBRM
Especifica que db2sqljbind genera módulos de petición de base de datos (DBRM) a partir de un perfil serializado y que db2sqljbind no realiza operaciones de vinculación remotas.

-genDBRM sólo se aplica a los programas que se deben ejecutar sobre servidores de bases de datos DB2 para z/OS.

-DBRMDir *nombre-directorio*
Cuando se especifica -genDBRM, -DBRMDir especifica el directorio local en el que db2sqljbind coloca los archivos DBRM generados. El valor por omisión es el directorio activo.

-DBRMdir sólo se aplica a los programas que se deben ejecutar sobre servidores de bases de datos DB2 para z/OS.

-tracefile *nombre-archivo*
Habilita el rastreo e identifica el archivo de salida para la información de rastreo. Esta opción debe especificarse solamente bajo la dirección del soporte de software de IBM.

-tracelevel
Si se especifica -tracefile, indica lo que se desea rastrear durante la ejecución de `db2sqljcustomize`. El valor por omisión es TRACE_SQLJ. Esta opción debe especificarse solamente bajo la dirección del soporte de software de IBM.

nombre-perfil-serializado|*nombre-archivo*`.grp`
Especifica los nombres de uno o varios perfiles serializados desde los que se ha vinculado el paquete. Un nombre de perfil serializado tiene el formato siguiente:

*nombre-programa*_SJProfile*NúmeroID*.ser

nombre-programa es el nombre del programa fuente de SQLJ, sin la extensión .sqlj. *n* es un entero entre 0 y *m-1*, donde *m* es el número de perfiles serializados generados por el conversor de SQLJ a partir del programa fuente de SQLJ.

Puede especificar nombres de perfiles serializados de una de las maneras siguientes:

- Liste los nombres en el mandato db2sqljcustomize. Si desea especificar varios nombres de perfiles serializados deben estar separados por espacios.
- Especifique los nombres de perfiles serializados, uno en cada línea, en un archivo con el nombre *nombre-archivo*.grp y especifique *nombre-archivo*.grp en el mandato db2sqljbind.

Si especifica más de un nombre de perfil serializado para vincular un único paquete de DB2 desde varios perfiles serializados, debe haber especificado los mismos nombres de perfiles serializados, en el mismo orden, cuando ejecutó db2sqljcustomize.

Si especifica uno o varios archivos *nombre-archivo*.grp, debe haber ejecutado db2sqljcustomize una vez con esa misma lista de archivos. El orden en el que se especifican los archivos en db2sqljbind debe ser igual que el orden en db2sqljcustomize.

No se puede ejecutar db2sqljcustomize en archivos individuales y agrupar estos archivos al ejecutar db2sqljbind.

Ejemplos

```
db2sqljbind -user richler -password mordecai
  -url jdbc:db2://server:50000/sample -bindoptions "EXPLAIN YES"
  pgmname_SJProfile0.ser
```

Notas de uso

Nombres de paquetes generados por db2sqljbind: los nombres de los paquetes creados por `db2sqljbind` son los nombres especificados utilizando los parámetros -rootpkgname o -singlepkgname al ejecutar `db2sqljcustomize`. Si no se especificó -rootpkgname o -singlepkgname, los nombres de paquetes son los primeros siete bytes del nombre del perfil, a los que se añade el carácter de nivel de aislamiento.

Valor DYNAMICRULES para db2sqljbind: La opción de vinculación DYNAMICRULES determina varios atributos de ejecución para el paquete DB2. Dos de esos atributos son el ID de autorización que se utiliza para comprobar la autorización y el calificador que se utiliza para los objetos no calificados. Para asegurar la autorización correcta para sentencias UPDATE y DELETE de posición ejecutadas dinámicamente en programas SQLJ, `db2sqljbind` siempre vincula paquetes DB2 mediante la opción DYNAMICRULES(BIND). No se puede modificar esta opción. La opción DYNAMICRULES(BIND) hace que las sentencias SET CURRENT SQLID y las sentencias SET CURRENT SCHEMA no tengan impacto en un programa SQLJ, ya que estas sentencias solamente afectan a las sentencias dinámicas vinculadas con los valores DYNAMICRULES distintos de BIND.

Con DYNAMICRULES(BIND), los nombres de tabla, vista, índice y alias no calificados en sentencias de SQL dinámico están calificados de forma implícita con el valor de la opción de vinculación QUALIFIER. Si no especifica QUALIFIER, DB2 utiliza el ID de autorización del propietario del paquete como calificador implícito. Si este comportamiento no es adecuado para el programa, podrá utilizar una de las técnicas siguientes para establecer el calificador correcto:

- Haga que las sentencias UDPATE y DELETE de posición se ejecuten estáticamente. Para ello, se puede utilizar la opción -staticpositioned YES de db2sqljcustomize o db2sqljbind si el cursor (iterador) de una sentencia UPDATE o DELETE de posición está en el mismo paquete que la sentencia UPDATE o DELETE de posición.
- Califique al completo los nombres de tabla de DB2 en las sentencias UPDATE y DELETE de posición.

Comportamiento de la opción de vinculación EXTENDEDINDICATOR: Si la opción de vinculación EXTENDEDINDICATOR no se especifica en la serie de opciones -bindoptions y el servidor de datos de destino soporta los indicadores ampliados, las operaciones de vinculación utilizan EXTENDEDINDICATOR(YES).

Si EXTENDEDINDICATOR(NO) se ha especificado explícitamente y la aplicación contiene la sintaxis de indicador ampliado, el comportamiento puede ser inesperado ya que IBM Data Server Driver para JDBC y SQLJ trata los indicadores ampliados como valores NULL.

db2sqljprint - Impresora de perfiles de SQLJ

db2sqljprint imprime el contenido de la versión personalizada de un perfil en forma de texto plano.

Autorización

Ninguno

Sintaxis del mandato

Parámetros del mandato

nombre-perfil
Especifica el nombre relativo o absoluto de un archivo de perfil SQLJ. Cuando un archivo SQLJ se convierte en un archivo fuente Java, la información sobre las operaciones de SQL que contiene se almacenan en archivos de recursos generados por SQLJ denominados perfiles. Los perfiles se identifican mediante el sufijo _SJProfileN (siendo N un entero) después del nombre del archivo de entrada original. Tienen la extensión `.ser`. Los nombres de los perfiles se pueden especificar con la extensión `.ser` o sin ésta.

Ejemplos

```
db2sqljprint pgmname_SJProfile0.ser
```

Apéndice A. Visión general de la información técnica de DB2

La información técnica de DB2 está disponible en diversos formatos a los que se puede acceder de varias maneras.

La información técnica de DB2 está disponible a través de las herramientas y los métodos siguientes:

- Centro de información de DB2
  - Temas (Tareas, concepto y temas de consulta)
  - Programas de ejemplo
  - Guías de aprendizaje
- Manuales de DB2
  - Archivos PDF (descargables)
  - Archivos PDF (desde el DVD con PDF de DB2)
  - Manuales impresos
- Ayuda de la línea de mandatos
  - Ayuda de mandatos
  - Ayuda de mensajes

Nota: Los temas del Centro de información de DB2 se actualizan con más frecuencia que los manuales en PDF o impresos. Para obtener la información más actualizada, instale las actualizaciones de la documentación cuando estén disponibles, o consulte el Centro de información de DB2 en ibm.com.

Puede acceder a información técnica adicional de DB2 como, por ejemplo, notas técnicas, documentos técnicos y publicaciones IBM Redbooks en línea, en el sitio ibm.com. Acceda al sitio de la biblioteca de software de gestión de información de DB2 en http://www.ibm.com/software/data/sw-library/.

Comentarios sobre la documentación

Agradecemos los comentarios sobre la documentación de DB2. Si tiene sugerencias sobre cómo podemos mejorar la documentación de DB2, envíe un correo electrónico a db2docs@ca.ibm.com. El personal encargado de la documentación de DB2 lee todos los comentarios de los usuarios, pero no puede responderlos directamente. Incluya ejemplos específicos siempre que sea posible para que podamos comprender mejor sus inquietudes. Si sus comentarios tratan sobre un archivo de ayuda o tema específico, especifique el título del tema y el URL.

No utilice esta dirección de correo electrónico para contactar con el Soporte al cliente de DB2. Si tiene un problema técnico con DB2 que no se resuelve en la documentación, póngase en contacto con su centro local de servicio técnico de IBM para obtener ayuda.

Visualización de ayuda de estado de SQL desde el procesador de línea de mandatos

Los productos DB2 devuelven 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 iniciar 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.

Acceso a diferentes versiones del Centro de información de DB2

La documentación correspondiente a otras versiones de los productos DB2 se encuentra en otros centros de información en ibm.com.

Acerca de esta tarea

Para los temas de la Versión 10.1 de DB2, el URL del *Centro de información de DB2 Information Center* es http://publib.boulder.ibm.com/infocenter/db2luw/v10r1.

Para los temas de la Versión 9.8 de DB2, el URL del *Centro de información de DB2* es http://publib.boulder.ibm.com/infocenter/db2luw/v9r8/.

Para los temas de DB2 Versión 9.7, el URL del *Centro de información de DB2* es http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/.

Para los temas de DB2 Versión 9.5, el URL del *Centro de información de DB2* es http://publib.boulder.ibm.com/infocenter/db2luw/v9r5.

Para los temas de DB2 Versión 9.1, el URL del *Centro de información de DB2* es http://publib.boulder.ibm.com/infocenter/db2luw/v9/.

Para los temas de DB2 Versión 8, vaya al URL del *Centro de información de DB2* en: http://publib.boulder.ibm.com/infocenter/db2luw/v8/.

Guías de aprendizaje de DB2

Las guías de aprendizaje de DB2 le ayudan a conocer diversos aspectos de productos DB2. Se proporcionan instrucciones paso a paso a través de lecciones.

Antes de comenzar

Puede ver la versión XHTML de la guía de aprendizaje desde el Centro de información en el sitio http://publib.boulder.ibm.com/infocenter/db2luw/v10r1/.

Algunas lecciones utilizan datos o código de ejemplo. Consulte la guía de aprendizaje para obtener una descripción de los prerrequisitos para las tareas específicas.

Guías de aprendizaje de DB2

Para ver la guía de aprendizaje, pulse el título.

pureXML
Configure una base de datos DB2 para almacenar datos XML y realizar operaciones básicas con el almacén de datos XML nativos.

Información de resolución de problemas de DB2

Existe una gran variedad de información para la resolución y determinación de problemas para ayudarle en la utilización de productos de base de datos DB2.

Documentación de DB2
Puede encontrar información sobre la resolución de problemas en la publicación *Troubleshooting and Tuning Database Performance* o en la sección sobre conceptos fundamentales sobre bases de datos del Centro de información de *DB2*, que contiene:

- Información sobre cómo aislar e identificar problemas con programas de utilidad y herramientas de diagnóstico de DB2.
- Soluciones a algunos de los problemas más comunes.
- Consejo para ayudarle a resolver problemas que podría encontrar en los productos de base de datos DB2.

Portal de Soporte de IBM
Consulte el portal de Soporte de IBM si tiene problemas y desea obtener ayuda para encontrar las causas y soluciones posibles. El sitio de soporte técnico tiene enlaces a las publicaciones más recientes de DB2, notas técnicas, Informes autorizados de análisis del programa (APAR o arreglos de defectos), fixpacks y otros recursos. Puede buscar en esta base de conocimiento para encontrar posibles soluciones a los problemas.

Acceda al portal de soporte de IBM en http://www.ibm.com/support/entry/portal/Overview/Software/Information_Management/DB2_for_Linux,_UNIX_and_Windows

Términos y condiciones

Los permisos para utilizar estas publicaciones se otorgan sujetos a los siguientes términos y condiciones.

Uso personal: Puede reproducir estas publicaciones para su uso personal, no comercial, siempre y cuando se mantengan los avisos sobre la propiedad. No puede distribuir, visualizar o realizar trabajos derivados de estas publicaciones, o de partes de las mismas, sin el consentimiento expreso de IBM.

Uso comercial: Puede reproducir, distribuir y visualizar estas publicaciones únicamente dentro de su empresa, siempre y cuando se mantengan todos los avisos sobre la propiedad. No puede realizar trabajos derivativos de estas publicaciones, ni reproducirlas, distribuirlas o visualizarlas, ni de partes de las mismas fuera de su empresa, sin el consentimiento expreso de IBM.

Excepto lo expresamente concedido en este permiso, no se conceden otros permisos, licencias ni derechos, explícitos o implícitos, sobre las publicaciones ni sobre ninguna información, datos, software u otra propiedad intelectual contenida en el mismo.

IBM se reserva el derecho de retirar los permisos aquí concedidos cuando, a su discreción, el uso de las publicaciones sea en detrimento de su interés o cuando, según determine IBM, las instrucciones anteriores no se cumplan correctamente.

El usuario no puede descargar, exportar o volver a exportar esta información, excepto si cumple por completo todas las leyes y regulaciones aplicables, incluidas las leyes y regulaciones de exportación de Estados Unidos.

IBM NO GARANTIZA EL CONTENIDO DE ESTAS PUBLICACIONES. LAS PUBLICACIONES SE PROPORCIONAN "TAL CUAL" Y SIN GARANTÍA DE NINGUNA CLASE, NI EXPLÍCITA NI IMPLÍCITA, INCLUYENDO PERO SIN LIMITARSE A LAS GARANTÍAS IMPLÍCITAS DE COMERCIALIZACIÓN, NO VULNERACIÓN E IDONEIDAD PARA UN FIN DETERMINADO.

Apéndice B. Avisos

Esta información ha sido desarrollada para productos y servicios que se ofrecen en Estados Unidos de América. La información acerca de productos que no son IBM se basa en la información disponible cuando se publicó este documento por primera vez y está sujeta a cambio.

Puede que IBM no ofrezca en otros países los productos, servicios o características que se explican en este documento. 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, puede utilizarse cualquier producto, programa o servicio que sea funcionalmente equivalente y que no infrinja ningún derecho de propiedad intelectual de IBM. Sin embargo, será 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 patente pendientes de aprobación que cubran los temas descritos 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 efectuar consultas relativas a la información de juego de caracteres de doble byte (DBCS), póngase en contacto con el departamento de propiedad intelectual de IBM de su país o envíe las consultas por correo a:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 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 o 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. El contenido de esos sitios web no forma parte del contenido del presente producto de IBM y la utilización de esos sitios web corre a cuenta y riesgo del usuario.

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
U59/3600
3600 Steeles Avenue East
Markham, Ontario L3R 9Z7
CANADÁ

Dicha información puede estar disponible, sujeta a los términos y condiciones apropiados, incluido en algunos casos el pago de una tarifa.

IBM proporciona el programa bajo licencia que se describe en este documento y todo el material bajo licencia disponible para el mismo en los términos del IBM Customer Agreement (Contrato de cliente de IBM), del IBM International Program License Agreement (Contrato internacional de programas bajo licencia de IBM) o de cualquier contrato equivalente entre las dos partes.

Los datos de rendimiento contenidos en este documento se obtuvieron en un entorno controlado. Por lo tanto, los resultados que se obtengan 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. Es más, puede que algunas mediciones sean estimaciones obtenidas por 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 tales productos y no puede confirmar la precisión de su rendimiento, su compatibilidad ni ningún otro aspecto relacionado con 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 lo más exhaustivamente posible, los ejemplos incluyen nombres de personas, empresas, marcas y productos. Todos

estos nombres son ficticios y cualquier parecido con nombres y direcciones que pudieran utilizar empresas reales es pura coincidencia.

LICENCIA DE COPYRIGHT:

Este manual contiene 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 del modo que considere adecuado sin previo pago a IBM, con el objeto de desarrollar, utilizar, comercializar o distribuir programas de aplicación de acuerdo con la interfaz de programación de aplicaciones para la plataforma operativa para la cual se han escrito los programas de ejemplo. Estos ejemplos no se han probado de forma exhaustiva bajo todas las condiciones. IBM, por tanto, no puede garantizar ni presuponer la fiabilidad, servicio o funcionamiento de dichos programas. Los programas de ejemplo se proporcionan "TAL CUAL", sin ningún tipo de garantía. IBM no se hará responsable de los daños derivados de la utilización que haga el usuario de los programas de ejemplo.

Cada copia de cualquier parte de estos programas de ejemplo o de cualquier trabajo que derive de éstos, debe incluir un aviso de copyright, tal como se indica a continuación:

© (*nombre de su empresa*) (*año*). Las partes de este código derivan de IBM Corp. Programas de ejemplo. © Copyright IBM Corp. *_entre el o los años_*. All rights reserved.

Marcas registradas

IBM, el logotipo de IBM e ibm.com son marcas registradas de International Business Machines Corp., que se han registrado en muchas otras jurisdicciones. Otros nombres de productos y servicios pueden ser marcas registradas de IBM o de otras empresas. Puede consultarse en línea una lista actualizada de las marcas registradas de IBM en la sección “Copyright and trademark information” en www.ibm.com/legal/copytrade.shtml.

Los siguientes términos son marcas registradas de otras empresas.

- Linux es una marca registrada de Linus Torvalds en los Estados Unidos o en otros países.
- Java y todos los logotipos y marcas registradas basados en Java son marcas registradas de Oracle o sus filiales.
- UNIX es una marca registrada de The Open Group en los Estados Unidos o en otros países.
- Intel, el logotipo de Intel, Intel Inside, el logotipo de Intel Inside, Celeron, Intel SpeedStep, Itanium y Pentium son marcas registradas de Intel Corporation o de sus empresas subsidiarias en Estados Unidos y en otros países.
- Microsoft, Windows, Windows NT y el logotipo de Windows son marcas registradas de Microsoft Corporation en los Estados Unidos o en otros países.

Otros nombres de empresas, productos o servicios utilizados en esta publicación pueden ser marcas registradas o marcas de servicio de terceros.

Índice

Caracteres Especiales

A

B

C

D

E

F

G

H

I

J

M

N

O

P

R

S

T

U

V

W

X

SC11-8065-01

Spine information:

IBM DB2 10.1 para Linux, UNIX y Windows

Desarrollo de aplicaciones Java