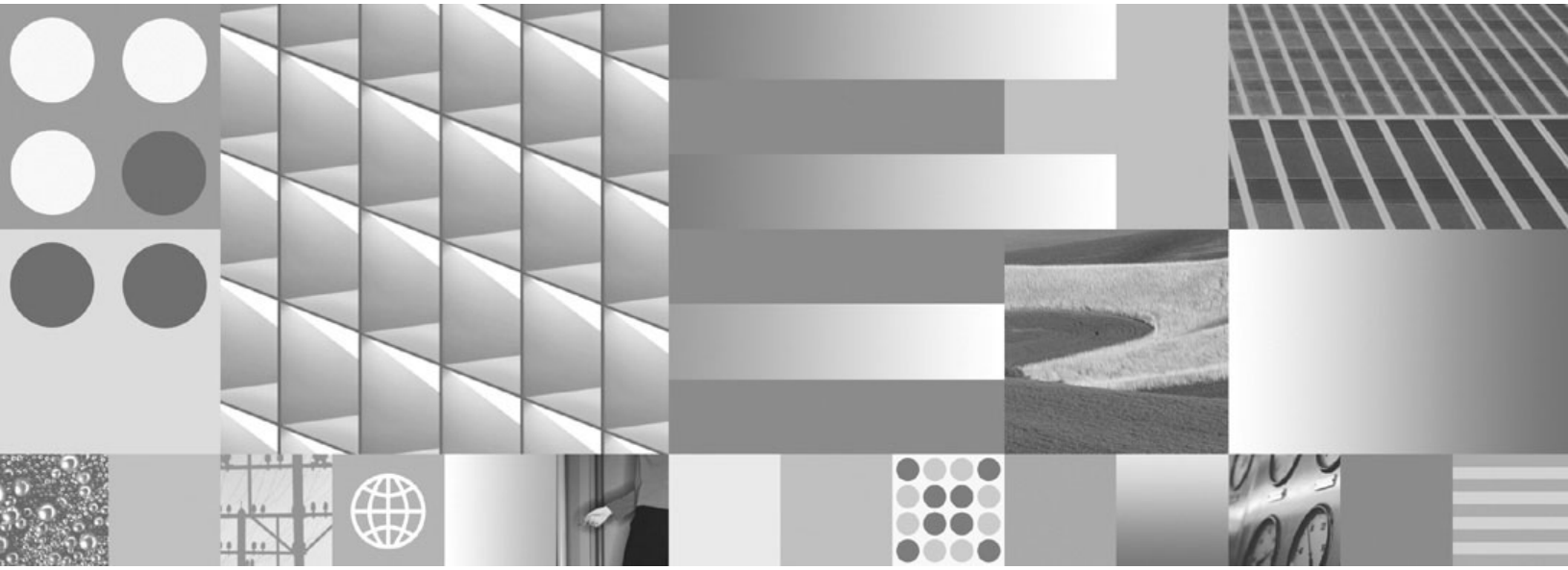
DB2 Versión 9.5
for Linux, UNIX, and Windows

Versión 9 Release 5

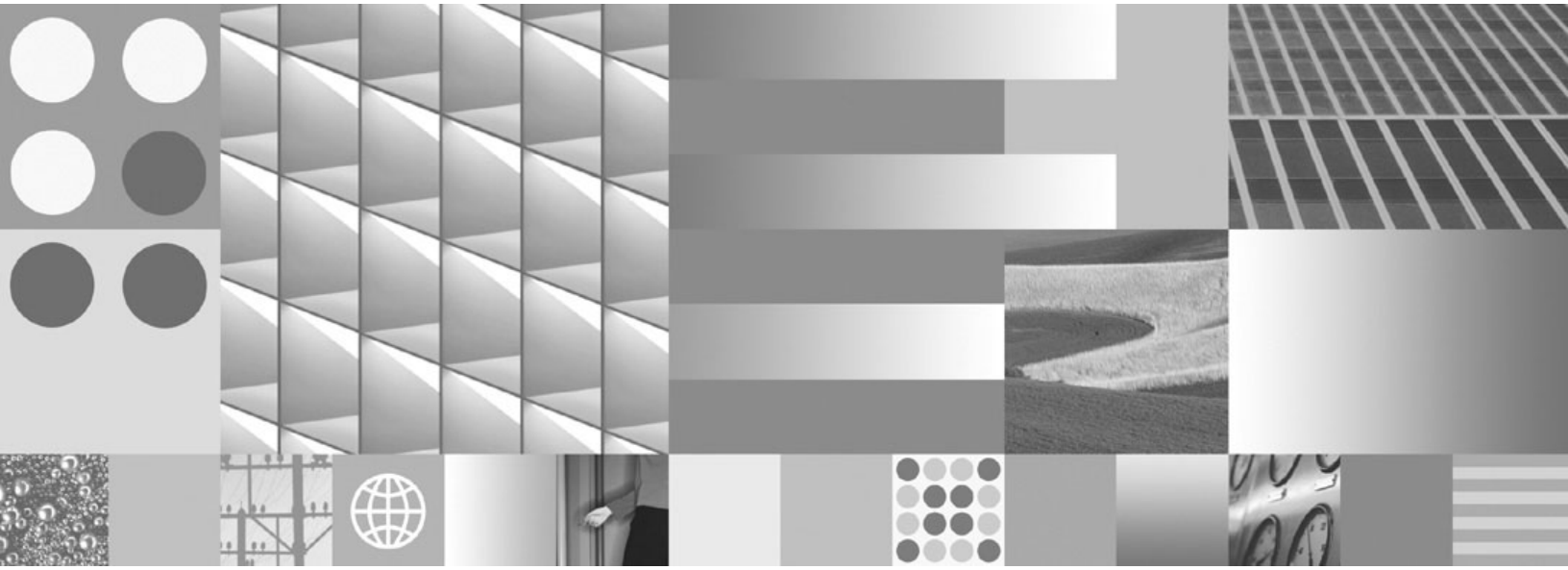


Desarrollo de aplicaciones Java

SC11-3501-00

DB2 Versión 9.5
for Linux, UNIX, and Windows

Versión 9 Release 5



Desarrollo de aplicaciones Java

SC11-3501-00

Nota

Antes de utilizar esta información y el producto al que da soporte, lea la información general contenida en el apartado Apéndice B, "Avisos", en la página 443.

Nota de edición

Esta publicación es la traducción del original inglés: DB2 Version 9.5 for Linux, UNIX, and Windows - Developing Java Applications, (SC23-5853-00).

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

- Para realizar pedidos en línea, vaya a IBM Publications Center ubicado en el sitio web www.ibm.com/shop/publications/order
- Para encontrar al representante de IBM de su localidad, vaya al IBM Directory of Worldwide Contacts en el sitio web 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).

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

Contenido

Acerca de este manual

Este manual describe el soporte de DB2 para Linux, UNIX y Windows de 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 DB2

El sistema de bases de datos DB2 proporciona soporte de controlador para aplicaciones cliente y applets que están escritos en Java utilizando JDBC, y para SQL incorporado para Java (SQLJ).

JDBC es una interfaz de programación de aplicaciones (API) que las aplicaciones de Java utilizan para acceder a las bases de datos relacionales. El soporte de DB2 para JDBC le permite escribir aplicaciones Java que acceden a datos DB2 locales o datos relacionales remotos situados en un servidor que es compatible con DRDA.

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.

En general, las aplicaciones Java utilizan JDBC para el SQL dinámico y SQLJ para el SQL estático. Sin embargo, debido a que SQLJ puede interaccionar con JDBC, un programa de aplicación puede utilizar JDBC y SQLJ dentro de la misma unidad de trabajo.

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 es compatible con el controlador de tipo 1.

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

Tipo 3
Son controladores que utilizan un cliente Java puro y se comunican con una base de datos utilizando un protocolo independiente de la base de datos. A continuación, la base de datos transmite las peticiones del cliente a la fuente de datos. El sistema de bases de datos DB2 no es compatible con 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 los controladores siguientes:

Nombre de controlador	Empaquetado como	Tipo de controlador
Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	db2java.zip	Tipo 2
IBM Data Server Driver para JDBC y SQLJ	• db2jcc.jar y sqlj.zip para JDBC 3.0, soporte • db2jcc4.jar y sqlj.zip para JDBC 4.0, soporte	Tipo 2 y Tipo 4

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

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

Existen dos versiones disponibles de IBM Data Server Driver para JDBC y SQLJ. IBM Data Server Driver para JDBC y SQLJ Versión 3.5 es compatible con JDBC 3.0. IBM Data Server Driver para JDBC y SQLJ Versión 4.0 es compatible con JDBC 3.0 y permite ejecutar algunas funciones de JDBC 4.0.

El IBM Data Server Driver para JDBC y SQLJ soporta estas funciones JDBC y SQLJ:
- Todos los métodos que se describen en las especificaciones de JDBC 3.0. Consulte ″Soporte de controlador para las API de JDBC″.
- Algunos métodos que están descritos en las especificaciones de JDBC 4.0, si instala IBM Data Server Driver para JDBC y SQLJ Versión 4.0.
- Interfaces de programación de aplicaciones SQLJ, tal como están definidas por 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.
- Funciones definidas por el usuario y procedimientos almacenados de Java (sólo IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 2).

  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).

Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows (controlador JDBC de DB2 de tipo 2) (en desuso)

El controlador JDBC de DB2 de tipo 2 permite que las aplicaciones Java realicen llamadas a DB2 mediante JDBC. Las llamadas al controlador JDBC de DB2 de tipo 2 se convierten en métodos nativos de Java. El controlador JDBC de DB2 de tipo 2 utiliza la interfaz de línea de mandatos (CLI) de DB2 para comunicarse con bases de datos DB2. Las aplicaciones Java que hacen uso de este controlador se deben ejecutar en un cliente DB2, a través del cual las peticiones JDBC fluyen hacia la base de datos DB2. Se debe instalar DB2 Connect para poder utilizar el controlador de aplicaciones JDBC de DB2 para acceder a fuentes de datos DB2 para i5/OS o fuentes de datos situadas en entornos de DB2 para z/OS.

El controlador JDBC de DB2 de tipo 2 es compatible con estas funciones de JDBC y SQLJ:

- La mayoría de los métodos que se describen en la especificación de JDBC 1.2 y algunos de los métodos que se describen en la especificación de JDBC 2.0.
- Sentencias de SQLJ que ejecutan operaciones equivalentes a todos los métodos JDBC
- Agrupación de conexiones
- Transacciones distribuidas
- Funciones definidas por el usuario y procedimientos almacenados de Java

El controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows no podrá utilizarse en releases futuros. Debe pues considerar la posibilidad de migrar hacia el IBM Data Server Driver para JDBC y SQLJ.

Compatibilidad del controlador JDBC y la base de datos DB2

Puede utilizar IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 4 para conectar con una base de datos DB2 perteneciente a un nivel de release diferente.

El 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 puede utilizar IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 4 desde IBM Data Server Driver para JDBC y SQLJ Versión 3.50, que se proporciona con DB2 Database para Linux, UNIX y Windows Versión 9.5, con una base de datos DB2 Versión 8.

El 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, se puede utilizar IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 4 desde IBM Data Server Driver para JDBC y SQLJ Versión 2.9.31, que se proporciona con DB2 Database para Linux, UNIX y Windows Versión 8 Fix Pack 12, con una base de datos DB2 Versión 9.5, si las aplicaciones bajo las que se ejecuta el controlador no contienen ninguna función de DB2 Versión 9.5.

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 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, 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 piensa ejecutar aplicaciones JDBC o SQLJ en el sistema, pero no prepararlas, necesita solamente un entorno de ejecución Java.
- Soporte de hebras nativas de JVM

  Todos los 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 algunos 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 algún programa SQLJ o JDBC utilizará IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 4 para conectar con un servidor DB2 para i5/OS, el sistema operativo OS/400 debe ser compatible con el sistema de codificación UTF-8 de Unicode. La tabla siguiente lista los PTF de OS/400 que son necesarios para dar soporte a UTF-8 de Unicode:

Tabla 1. PTF de OS/400 para el soporte de UTF-8 de Unicode

Versión OS/400	Números de PTF
V5R3 o posterior	Ninguno (el soporte está incluido)
V5R2	SI06541, SI06796, SI07557, SI07564, SI07565, SI07566 y SI07567
V5R1	SI06308, SI06300, SI06301, SI06302, SI06305, SI06307 y SI05872

- Soporte de Java a clientes y servidores HP-UX

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

  “Instalación especial para ejecutar rutinas Java en el entorno HP-UX” en el manual *Desarrollo de aplicaciones Java*

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

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:

   - 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 y 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 y funciones posteriores, así como JDBC 3.0 y funciones anteriores.**

     sqlj.zip
     sqlj.zip se utiliza para preparar y ejecutar programas SQLJ.

   - 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 el archivo db2jcc.jar. Si está utilizando el archivo db2jcc4.jar, debe modificar manualmente la variable CLASSPATH. Cambie db2jcc.jar por db2jcc4.jar en CLASSPATH.

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

   - Instala archivos de licencia de IBM Data Server Driver para JDBC y SQLJ y modifica la variable CLASSPATH para incluir esos archivos.

     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 2. Archivos de licencia del 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 i5/OS	Todos los productos de DB2 Connect

     Los archivos de licencia no son necesarios para las conexiones con bases de datos DB2 Database para Linux, UNIX y Windows, Cloudscape o IBM Informix Dynamic Server (IDS) realizadas desde IBM Data Server Driver para JDBC y SQLJ Versión 3.50 o posterior.

   - Instala las bibliotecas nativas de IBM Data Server Driver para JDBC y SQLJ para poder utilizar IBM Data Server Driver para conectividad JDBC y SQLJ 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

libdb2jcct2.sl
Para HP-UX sobre PA-RISC

db2jcct2.dll
Para Windows

2. Personalice las propiedades de configuración del controlador si cualquiera de los valores por omisión no son adecuados. Consulte "Personalización de las propiedades de configuración del controlador IBM DB2 para JDBC y SQLJ" en el manual *Desarrollo de aplicaciones Java* para ver detalles.
3. Configure TCP/IP.

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

   - Aplicaciones JDBC o SQLJ que utilizan el IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 4.
   - Aplicaciones JDBC o SQLJ que utilizan el IBM Data Server Driver para conectividad JDBC y SQLJ 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 de servicios:

      ```
      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, compruebe que la variable de entorno DB2_USE_DB2JCCT2_JROUTINE no esté establecida, o que esté establecida en su valor por omisión: `YES`, `yes`, `ON`, `on`, `TRUE`, `true` o 1 en esos servidores de bases de datos. Este valor indica que los procedimientos almacenados Java se ejecutan bajo IBM Data Server Driver para JDBC y SQLJ.

   Si necesita ejecutar procedimientos almacenados bajo el controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows, establezca la variable de entorno DB2_USE_DB2JCCT2_JROUTINE en `OFF`.
5. 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.

6. 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.

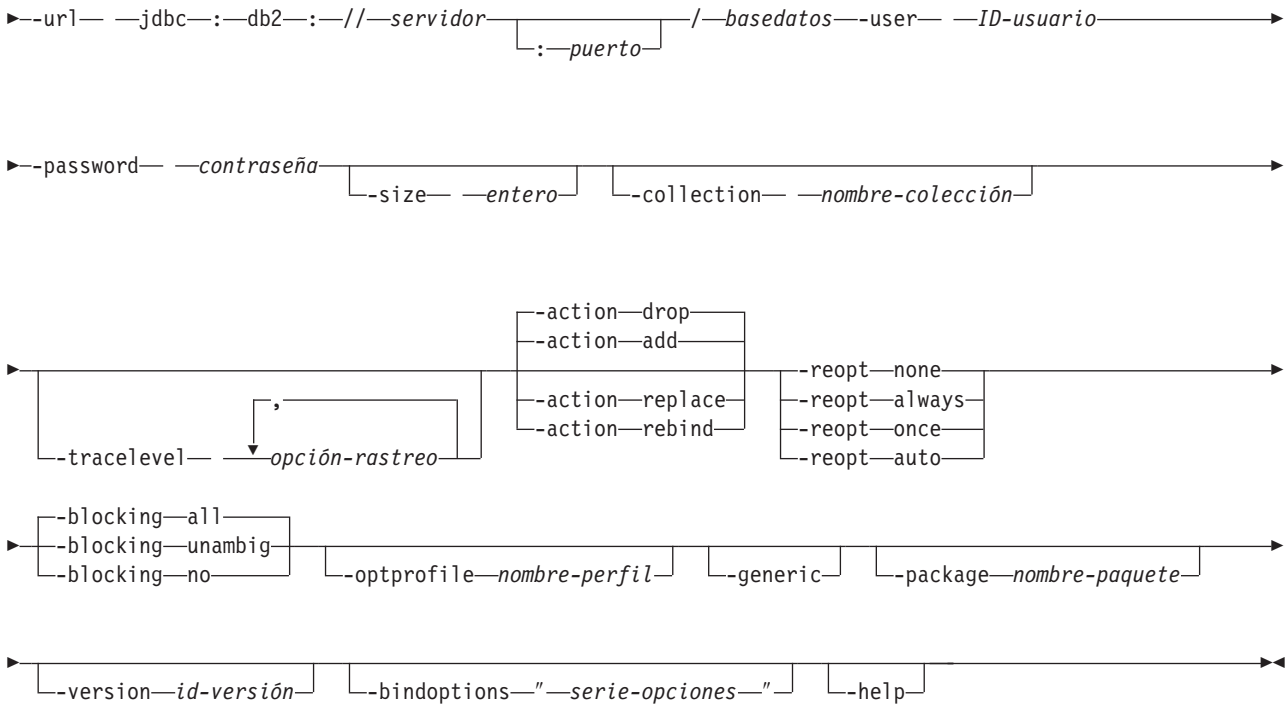
   Estos pasos son necesarios para garantizar que la aplicación de llamada recibe correctamente los valores de fecha y hora.
7. Si tiene intención de acceder a servidores de bases de datos DB2 para z/OS con sus aplicaciones Java, siga las instrucciones de “Instalación especial para acceder a servidores DB2 para z/OS desde programas Java programas” en el manual *Desarrollo de aplicaciones Java* .

Programa de utilidad DB2Binder

El programa de utilidad DB2Binder vincula los paquetes de DB2 utilizados en el servidor de bases 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

```
►►—java—com.ibm.db2.jcc.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 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 debe 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, el valor de *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"
```

-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 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 de ubicación del servidor de bases 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 si los paquetes del IBM Data Server Driver para JDBC y SQLJ pueden sustituirse.

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 eliminar algunos o todos los paquetes de IBM Data Server Driver para JDBC y SQLJ. El número de paquetes depende del parámetro -size.

La opción -action drop solamente es aplicable si el servidor de bases 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 eliminan. 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 elimina 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 eliminan 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 bases de datos ejecutando `com.ibm.db2.jcc.DB2Binder` varias veces, especificando un valor diferente para -collection cada vez. Durante la ejecución, seleccione una instancia del 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 bases de datos DB2 para z/OS determinan las vías de acceso en tiempo de ejecución. Esta opción sólo es válida para las conexiones a servidores de bases de datos DB2 para z/OS. Esta opción no se envía al servidor de bases de datos si no se especifica. En este caso, el servidor de bases de datos determina el comportamiento de reoptimización.

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 en tiempo de ejecución.

-blocking
Especifica el tipo de bloqueo de fila para cursores.

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

UNAMBIG
Para los cursores que están especificados por la cláusula FOR READ ONLY, se produce el bloqueo.

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 el bloqueo del cursor. Los cursores *ambiguos* no se bloquearán.

NO No se produce bloqueo 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 bases de datos DB2 Database para Linux, UNIX y Windows.

-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.

-bindoptions
Especifica una serie de caracteres que está delimitada por comillas. La serie de caracteres está formada uno más pares parámetro-valor que representan opciones para revincular un paquete de usuario.

Los parámetros y valores posibles son:

bindObjectExistenceRequired
Especifica si el servidor de bases 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 bases 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 bases de datos procesa las sentencias de SQL dinámico utilizando el comportamiento de invocación. En otro caso, el servidor de bases 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 bases de datos procesa las sentencias de SQL dinámico utilizando el comportamiento de invocación. En otro caso, el servidor de bases 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 bases de datos procesa las sentencias de SQL dinámico utilizando el comportamiento de definición. En otro caso, el servidor de bases 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 bases de datos procesa las sentencias de SQL dinámico

utilizando el comportamiento de definición. En otro caso, el servidor de bases 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 del paquete.

Esta opción corresponde a la opción de vinculación OWNER.

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.

Códigos de retorno de DB2Binder

DB2Binder devuelve uno de los códigos de retorno siguientes:

Tabla 3. Códigos de retorno de DB2Binder

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.
-104	No se ha especificado el valor de -action.

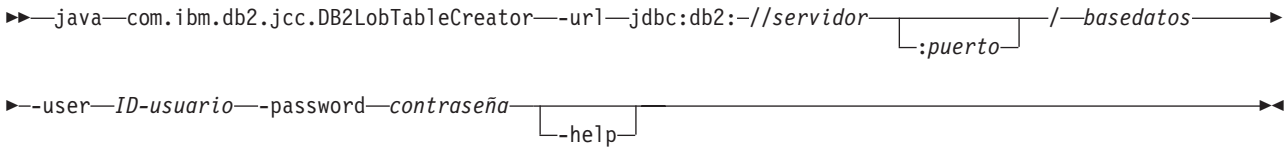
Tabla 3. Códigos de retorno de DB2Binder (continuación)

Código de retorno	Significado
-105	No se ha especificado el valor de -blocking.
-106	No se ha especificado el valor de -collection.
-107	No se ha especificado el valor de -dbprotocol.
-108	No se ha especificado el valor de -keepdynamic.
-109	No se ha especificado el valor de -owner.
-110	No se ha especificado el valor de -reopt.
-111	No se ha especificado el valor de -size.
-112	No se ha especificado el valor de -tracelevel.
-113	No se ha especificado el valor de -optprofile.
-200	No se han especificado opciones de vinculación válidas.
-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 bases de datos de destino.
-308	El valor de -keepdynamic no es compatible con el servidor de bases de datos de destino.
-310	El valor de -reopt no es compatible con el servidor de bases de datos de destino.
-313	El valor de -optprofile no es compatible con el servidor de bases de datos de destino.
-401	No se ha encontrado la clase `Binder`.
-402	La conexión con el servidor de bases de datos ha fallado.
-403	La recuperación de DatabaseMetaData para el servidor de bases 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 del 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 del IBM Data Server Driver para JDBC y SQLJ tiene este formato:

```
propiedad=valor
```

propiedad puede tener uno o varios de los formatos siguientes:
- db2.jcc.override.*nombre_propiedad*
- db2.jcc.*nombre_propiedad*
- db2.jcc.default.*nombre_propiedad*

Si la propiedad de configuración comienza por db2.jcc.override, esta propiedad se aplica a todas las conexiones y prevalece sobre cualquier propiedad de Connection o DataSource con el mismo valor *nombre_propiedad*. Si la propiedad de configuración comienza por db2.jcc o db2.jcc.default, el valor de la propiedad de configuración es un valor por omisión. Los valores de las propiedades de Connection o DataSource prevalecen sobre ese valor.

Puede establecer las propiedades de configuración de estas formas:

- Establezca las propiedades de configuración como propiedades del sistema Java. Estos valores prevalecen sobre cualquier otro.

  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. El 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 el 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.

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

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 4. PTF para procedimientos almacenados de DB2 para z/OS

DB2 para z/OS	Números de PTF o APAR
Versión 7	UQ72083, UQ93889, UK21848
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 el IBM Data Server Driver para JDBC y SQLJ. Consulte "Programa de utilidad DB2Binder" en la página 8 para conocer detalles.
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
     - 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. Consulte "Programa de utilidad DB2LobTableCreator" en la página 16 para conocer detalles.

4. Habilite el soporte de Unicode para servidores OS/390 y z/OS.

   Si cualquiera de los programas SQLJ o JDBC van a utilizar el IBM Data Server Driver para conectividad JDBC y SQLJ 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 y II13049 de información contienen datos adicionales.

5. Si tiene previsto utilizar el IBM Data Server Driver para conectividad JDBC y SQLJ 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. Consulte "DB2T4XAIndoubtUtil para transacciones distribuidas con DB2 UDB para los servidores OS/390 y z/OS Versión 7" para conocer detalles.

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 conectividad JDBC y SQLJ 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

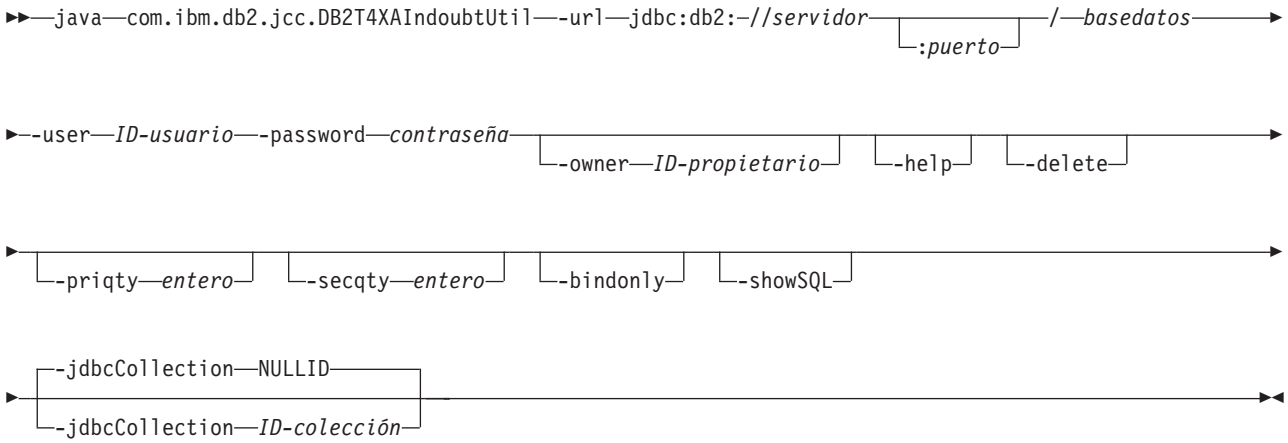
Debe crear y descartar los paquetes T4XAIN01, T4XAIN02, T4XAIN03 y T4XAIN04 sólo mediante la ejecución de DB2T4XAIndoubtUtil. Puede crear y descartar SYSTEM.INDOUBT y su índice manualmente, pero se recomienda que emplee el programa de utilidad. Consulte las Notas de utilización de DB2T4XAIndoubtUtil 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 tabla 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 tabla donde reside SYSIBM.INDOUBT debe ser mayor que el número máximo de transacciones dudosas que pueden existir en un momento cualquiera. Por ejemplo, para una 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 tabla 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 del 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
```

Instalación especial para ejecutar rutinas Java en el entorno HP-UX

Para el sistema operativo HP-UX sobre procesadores PA-RISC, existen requisitos adicionales para ejecutar procedimientos almacenados Java y una función definida por el usuario.

Además de los requisitos descritos en "Instalación de IBM Data Server Driver para JDBC y SQLJ", debe ejecutar los pasos necesarios siguientes:

1. Habilite la herramienta db2hpjv emitiendo los mandatos siguientes en la línea de mandatos:

   ```
   db2hpjv -e
   db2stop
   db2start
   ```

   Si necesita inhabilitar db2hpjv, ejecute estos mandatos:

   ```
   db2hpjv -d
   db2stop
   db2start
   ```

   Si necesita inhabilitar db2hpjv, emita estos mandatos: Java **debe** estar instalado en el sistema operativo para poder emitir `db2hpjv -e`. DB2 Database para Linux, UNIX y Windows no puede ejecutarse en HP-UX si está habilitado el soporte de rutina Java y Java no está instalado en el sistema operativo.
2. Proporcione acceso al editor de enlaces de ejecución de HP-UX para las bibliotecas compartidas de Java.

   Para ejecutar procedimientos almacenados Java o una función definida por el usuario, el editor de enlaces de ejecución de HP-UX debe poder acceder a determinadas bibliotecas compartidas de Java, y el sistema DB2 debe poder cargar esas bibliotecas y la JVM. Debido a que el programa que realiza esta carga se ejecuta con privilegios setuid, **solamente** buscará las bibliotecas dependientes en `/usr/lib/pa20_64`. Para crear un acceso a las bibliotecas compartidas de Java, elija uno de los métodos siguientes:

- Cree enlaces simbólicos con las bibliotecas compartidas de Java. Para ello, inicie sesión como root y emita los mandatos siguientes para crear enlaces simbólicos en las bibliotecas compartidas de Java:

  ```
  ln -s /opt/java1.4/jre/lib/PA_RISC2.0W/*.sl    /usr/lib/pa20_64
  ln -s /opt/java1.4/jre/lib/PA_RISC2.0W/hotspot/*.sl   /usr/lib/pa20_64
  ```

  Estos mandatos crean enlaces simbólicos con las bibliotecas siguientes:

  ```
  /opt/java1.4/jre/lib/PA_RISC2.0W/libnet.sl
  /opt/java1.4/jre/lib/PA_RISC2.0W/libzip.sl
  /opt/java1.4/jre/lib/PA_RISC2.0W/librmi.sl
  /opt/java1.4/jre/lib/PA_RISC2.0W/libnio.sl
  /opt/java1.4/jre/lib/PA_RISC2.0W/libverify.sl
  /opt/java1.4/jre/lib/PA_RISC2.0W/libmlib_image.sl
  /opt/java1.4/jre/lib/PA_RISC2.0W/libhprof.sl
  /opt/java1.4/jre/lib/PA_RISC2.0W/libjaas_unix.sl
  /opt/java1.4/jre/lib/PA_RISC2.0W/libawt.sl
  /opt/java1.4/jre/lib/PA_RISC2.0W/libcmm.sl
  /opt/java1.4/jre/lib/PA_RISC2.0W/libdcpr.sl
  /opt/java1.4/jre/lib/PA_RISC2.0W/libdt_socket.sl
  /opt/java1.4/jre/lib/PA_RISC2.0W/libfontmanager.sl
  /opt/java1.4/jre/lib/PA_RISC2.0W/libioser12.sl
  /opt/java1.4/jre/lib/PA_RISC2.0W/libmawt.sl
  /opt/java1.4/jre/lib/PA_RISC2.0W/libjsound.sl
  /opt/java1.4/jre/lib/PA_RISC2.0W/libjava.sl
  /opt/java1.4/jre/lib/PA_RISC2.0W/libjawt.sl
  /opt/java1.4/jre/lib/PA_RISC2.0W/libjcov.sl
  /opt/java1.4/jre/lib/PA_RISC2.0W/libjcpm.sl
  /opt/java1.4/jre/lib/PA_RISC2.0W/libjdwp.sl
  /opt/java1.4/jre/lib/PA_RISC2.0W/libjpeg.sl
  /opt/java1.4/jre/lib/PA_RISC2.0W/hotspot/libjsig.sl
  /opt/java1.4/jre/lib/PA_RISC2.0W/hotspot/libjvm.sl
  ```

- Añada los directorios /opt/java1.4/jre/lib/PA_RISC2.0W y opt/java1.4/jre/lib/PA_RISC2.0W/hotspot al archivo /etc/dld.sl.conf **y** al entorno SHLIB_PATH.

3. Dé acceso al enlazador de tiempo de ejecución de HP-UX a las bibliotecas compartidas de Java.

   Si el servidor DB2 no puede encontrar las bibliotecas compartidas de Java cuando ejecuta una rutina Java, produce un error -4300.

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 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 la conectividad de tipo 2
    // de IBM Data Server Driver para JDBC y SQLJ,
    // args[0] puede ser MVS1DB2M. Para la
    // conectividad de tipo 4, args[0] podría ser
    // stlmvs1:10110/MVS1DB2M.

    if (args.length==0)
    {
      System.err.println ("Valor no válido. Primer argumento añadido a "+
       "jdbc:db2: debe especificar un URL válido.");
      System.exit(1);
    }
    url = urlPrefix + args[0];
```

```
    try
    {
      // Cargar el controlador
      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);                             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
```

Nota para la Figura 1 en la página 25:

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″.
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 (cargar el controlador JDBC) no es necesario si utiliza JDBC 4.0.
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.

El 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 Dynamic Server (IDS) 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 conectividad JDBC y SQLJ de tipo 2.

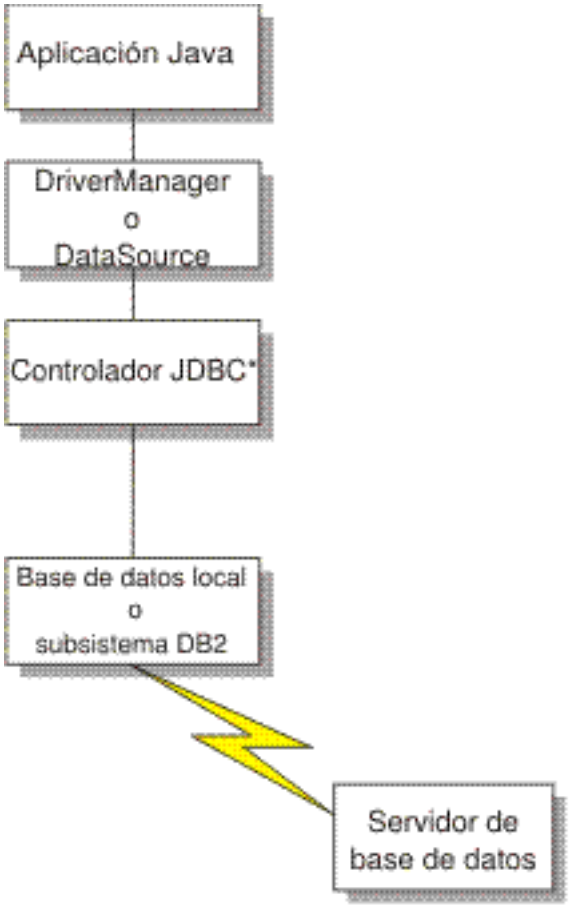


Figura 2. Flujo de una aplicación Java para IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 2

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

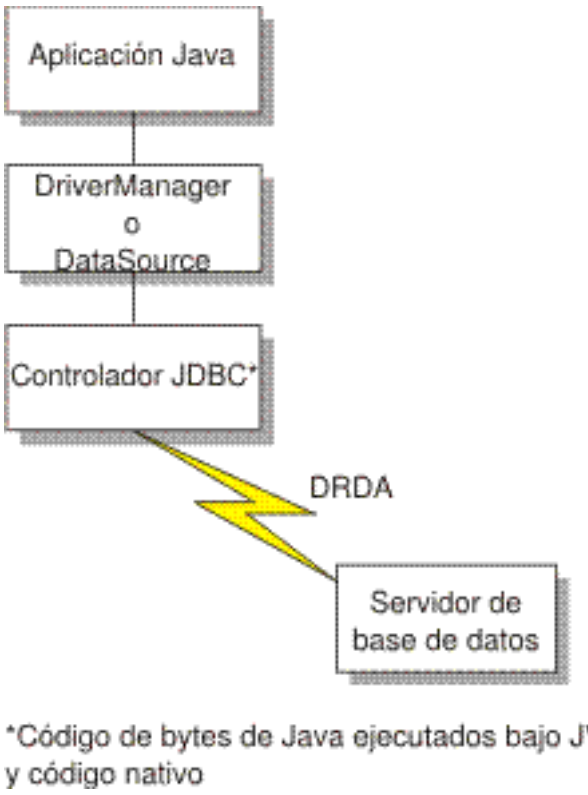


Figura 3. Flujo de una aplicación Java para IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 4

Conexión de aplicaciones DB2 a una fuente de datos utilizando la interfaz DriverManager con el controlador JDBC de DB2 de tipo 2

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

Primero la aplicación Java carga el controlador JDBC invocando el método Class.forName. Después de cargar el controlador, la aplicación conecta con una fuente de datos invocando el método DriverManager.getConnection.

Para el controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows (controlador JDBC de DB2 de tipo 2), cargue el controlador invocando el método Class.forName con el argumento siguiente:

```
COM.ibm.db2.jdbc.app.DB2Driver
```

El código siguiente muestra la carga del controlador JDBC de DB2 de tipo 2:

```
try {
  // Cargar el Controlador JDBC de DB2 de tipo 2 con DriverManager
  Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");
} catch (ClassNotFoundException e) {
    e.printStackTrace();
}
```

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

Después de cargar el controlador, se conecta a la fuente de datos invocando el método DriverManager.getConnection. Puede utilizar uno de los formatos siguientes de getConnection:

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

El argumento *url* representa una fuente de datos.

Para el Controlador JDBC de DB2 de tipo 2, especifique un URL de la forma siguiente:

Sintaxis de un URL para el controlador JDBC de DB2 de tipo 2:

Los elementos del URL tienen los significados siguientes:

jdbc:db2:
jdbc:db2: indica que la conexión es con una fuente de datos DB2.

basedatos
Es un alias de base de datos. El alias hace referencia a la entrada del catálogo de base de datos DB2 contenido en el cliente DB2.

El argumento *info* es un objeto de tipo java.util.Properties que contiene un conjunto de propiedades de controlador correspondientes a la conexión. Especificar el argumento *info* es una alternativa a especificar series de caracteres *propiedad=valor* en el URL.

Especificación de un ID de usuario y contraseña para una conexión: existen varias formas de especificar un ID de usuario y una contraseña para una conexión:

- Utilice el formato del método getConnection en el que se especifica el *usuario* y *contraseña*.
- Utilice la modalidad del método getConnection donde se especifica *info*, después de establecer la propiedades correspondientes al usuario y la contraseña en un objeto java.util.Properties.

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

```
String url = "jdbc:db2:toronto";
                                        // Definir URL para fuente de datos
String user = "db2adm";
String password = "db2adm";
Connection con = DriverManager.getConnection(url, user, password);
                                        // Crear conexión
```

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

```
Properties properties = new Properties(); // Crear objeto Properties
properties.put("user", "db2adm");  // Definir ID de usuario para conexión
properties.put("password", "db2adm");  // Definir contraseña para conexión
String url = "jdbc:db2:toronto";
```

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

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.

Estos son los pasos para establecer una conexión:

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

   Si está utilizando JDBC 4.0, no es necesario que cargue explícitamente 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 del 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 conectividad JDBC y SQLJ 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 la *contraseña*.
- Utilice la modalidad del método getConnection donde se especifica *info*, después de establecer la propiedades correspondientes al usuario y la contraseña en un objeto java.util.Properties.

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
```

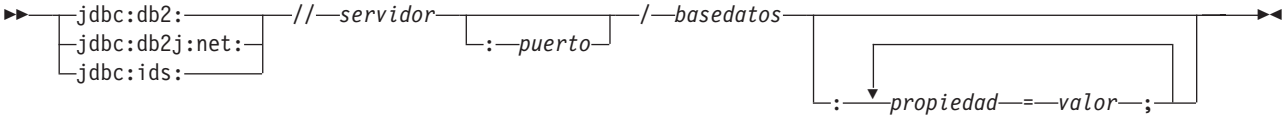
Ejemplo: establecimiento de una conexión y especificación del ID de usuario y la contraseña en un objeto java.util.Properties:

```
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 conectividad JDBC y SQLJ 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 conectividad JDBC y SQLJ de tipo 4Sintaxis del URL



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

Los elementos del URL tienen los significados siguientes:

jdbc:db2: o jdbc:db2j:net:
El significado de la porción inicial del URL es el siguiente:

jdbc:db2:
Indica que la conexión es con un servidor 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 Dynamic Server (IDS), 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 IDS. jdbc:informix-sqli: también indica que la conexión es con una fuente de datos IDS, 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. El IBM Data Server Driver para JDBC y SQLJ no convierte en mayúsculas para el IBM Data Server Driver para conectividad JDBC y SQLJ 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, todos los caracteres de la *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 IDS, *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 en blanco dentro de la lista de pares propiedad-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

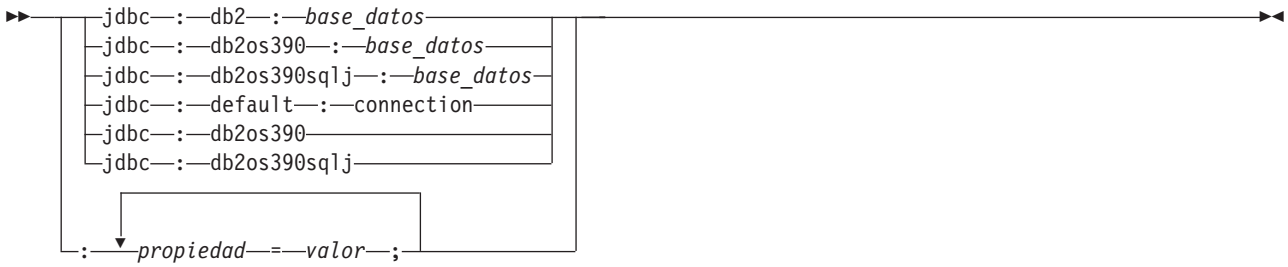
embargo, se puede crear una serie URL que incluya com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL, y asignar la serie URL a una variable String. A continuación, se puede utilizar la variable String en el parámetro *url*.

```
String url =
 "jdbc:db2://sysmvs1.stl.ibm.com:5021" +
 "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 conectividad JDBC y SQLJ 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 conectividad JDBC y SQLJ de tipo 2Sintaxis del URL



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

Los componentes del URL tienen los significados siguientes:

jdbc:db2: o jdbc:db2j:net:
Los significados del elemento inicial del URL son los siguientes:

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

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

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

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.

- Si la conexión es con un servidor DB2 para z/OS o un servidor DB2 para i5/OS, todos los caracteres de *basedatos* deben estar en mayúsculas.

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 en blanco dentro de la lista de pares propiedad-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" +
 "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.

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:

1. Consulte al administrador del sistema para obtener lógico de la fuente de datos con la que desee 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 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 que aparecen a la derecha de algunas sentencias corresponden a los 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 el 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

El 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 5. Comparación de IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 2 y IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 4

Función	Soporte de IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 2	Soporte de IBM Data Server Driver para conectividad JDBC y SQLJ 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 el IBM Data Server Driver para conectividad JDBC y SQLJ 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 el IBM Data Server Driver para conectividad JDBC y SQLJ 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.

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 conectividad JDBC y SQLJ de tipo 2 o IBM Data Server Driver para conectividad JDBC y SQLJ 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 conectividad JDBC y SQLJ de tipo 2 o IBM Data Server Driver para conectividad JDBC y SQLJ 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 conectividad JDBC y SQLJ de tipo 4.

El controlador JDBC de DB2 de tipo 2 proporciona las siguientes implementaciones de DataSource:

- COM.ibm.db2.jdbc.DB2DataSource, que está habilitado para la agrupación de conexiones. Con esta implementación; la agrupación de conexiones se maneja internamente y es transparente para la aplicación.
- COM.ibm.db2.jdbc.DB2XADataSource, que no tiene soporte interno para las transacciones distribuidas y la agrupación de conexiones. Con esta implementación, debe gestionar usted mismo las transacciones distribuidas y la agrupación de conexiones, ya sea grabando su propio código o utilizando una herramienta como WebSphere Application Server.

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 API de ampliación
                           // 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.

Nota	Descripción
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

javax.transaction
Contiene soporte de JDBC para transacciones distribuidas para el controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows (controlador JDBC de DB2 de tipo 2).

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

COM.ibm.db2.jdbc
Contiene la implementación del controlador JDBC de DB2 de tipo 2.

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.

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:

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

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
```

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.

Interfaces JDBC para ejecutar SQL

Puede ejecutar sentencias de SQL dentro de un programa SQL típico para insertar, actualizar, suprimir o fusionar datos de tablas, recuperar datos de tablas o llamar 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 el 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 el 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.

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 el 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:

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.

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.

Con el 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:
- Integer
- Double
- Decimal
- Character
- Hexadecimal
- Graphic

Para las llamadas a procedimientos almacenados que residen en fuentes de datos IBM Informix Dynamic Server, 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:

1. Invoque el método Connection.prepareStatement para crear un objeto PreparedStatement.
2. Invoque métodos PreparedStatement.setXXX para pasar valores a las variables.
3. Invoque el método PreparedStatement.executeUpdate para actualizar la tabla con los valores variables.
4. Invoque el método PreparedStatement.close para cerrar el objeto PreparedStatement cuando termine de utilizar ese objeto.

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

```
Connection con;
PreparedStatement pstmt;
int numUpd;
...
pstmt = con.prepareStatement(
  "UPDATE EMPLOYEE SET PHONENO=? WHERE EMPNO=?");
                                // Crear un objeto PreparedStatement       1
pstmt.setString(1,"4657");      // Asignar 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 en la página 44. 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 o DELETE, 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 tabla segmentado.
  - 0, si la operación no afecta a ninguna fila.
  - -1, si la operación es una supresión masiva sobre un espacio de tabla 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*.

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

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

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

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, el 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:

1. Invoque el método createStatement para crear un objeto Statement.
2. 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.
3. Invoque el método executeBatch para ejecutar las sentencias con todos los conjuntos de parámetros.
4. Compruebe si se han producido errores. 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.
   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, el IBM Data Server Driver para JDBC y SQLJ añade un método commit al final del proceso por lotes.

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 prepStmt = con.prepareStatement(
    "UPDATE DEPT SET MGRNO=? WHERE DEPTNO=?");           1
  prepStmt.setString(1,mgrnum1);                         2a
  prepStmt.setString(2,deptnum1);
  prepStmt.addBatch();                                   2b

  prepStmt.setString(1,mgrnum2);
  prepStmt.setString(2,deptnum2);
  prepStmt.addBatch();
  int [] numUpdates=prepStmt.executeBatch();             3
  for (int i=0; i < numUpdates.length; i++) {            4a
    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");
  }
  con.commit();                                          4b
} catch(BatchUpdateException b) {
  // process BatchUpdateException
}
```

Figura 11. Realización de una actualización de proceso por lotes

Obtención de información acerca de parámetros de PreparedStatement mediante métodos ParameterMetaData

El 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.

Los métodos de ParameterMetaData proporcionan los tipos de información siguientes:

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

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

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

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 12. 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.

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 el 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:

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.

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 13. 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.

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 el 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 Statement.execute. Consulte "Recuperación de varios conjuntos de resultados a partir de un procedimiento almacenado en una aplicación JDBC" para obtener más información.

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:

1. Invoque el método Connection.prepareStatement para crear un objeto PreparedStatement.
2. Invoque métodos PreparedStatement.setXXX para pasar valores a los parámetros de entrada.
3. Invoque el método PreparedStatement.executeQuery para obtener la tabla de resultados de la sentencia SELECT en un objeto ResultSet.
4. En un bucle, posicione el cursor utilizando el método ResultSet.next y recupere datos de cada columna de la fila actual del objeto ResultSet utilizando métodos getXXX.
5. Invoque el método ResultSet.close para cerrar el objeto ResultSet.
6. Invoque el método PreparedStatement.close para cerrar el objeto PreparedStatement cuando termine de utilizar ese objeto.

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 14. 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.

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

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. Difunda el objeto PreparedStatement a un objeto DB2PreparedStatement.
4. Invoque el método DB2PreparedStatement.executeBatch para ejecutar la sentencia con todos los conjuntos de parámetros.
5. Compruebe si se han producido errores.

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.

```
try {
...
  PreparedStatement prepStmt = con.prepareStatement(
    "SELECT EMPNO FROM EMPLOYEE WHERE EMPNO=?")          1
  prepStmt.setString(1,empnum1);                         2a
  prepStmt.addBatch();                                   2b

  prepStmt.setString(1,empnum2);
  prepStmt.addBatch();
  ((com.ibm.db2.jcc.DB2PreparedStatement)prepStmt).executeDB2QueryBatch();
```

```
} catch(BatchUpdateException b) {                    3,4
  // process BatchUpdateException                    5
}
```

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

No puede suponer que conoce 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.

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

Los métodos de ResultSetMetaData proporcionan los tipos de información siguientes:

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

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

1. Invoque el método getMetaData para el objeto ResultSet para crear un objeto ResultSetMetaData.
2. Invoque el método getColumnCount para determinar el número de columnas del conjunto de resultados.
3. Para cada columna del 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.

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

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

El 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.

Para las fuentes de datos de IBM Informix Dynamic Server, los cursores no pueden ser actualizables.

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 6. 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 Dynamic Server
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.

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. Para cambiar los atributos de capacidad de desplazamiento, actualización y retención para un ResultSet, siga estos pasos:

1. Si la sentencia SELECT por la que se define el conjunto de resultados no tiene parámetros de entrada, invoque el método createStatement para crear un objeto Statement. En otro caso, invoque el método prepareStatement para crear un objeto PreparedStatement. 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);
```

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

Tabla 7. Combinaciones válidas de valores de resultSetType y resultSetConcurrency para conjuntos de resultados desplazables

Valor de *resultSetType*	**Valor de** *resultSetConcurrency*
TYPE_FORWARD_ONLY	CONCUR_READ_ONLY
TYPE_FORWARD_ONLY	CONCUR_UPDATABLE
TYPE_SCROLL_INSENSITIVE	CONCUR_READ_ONLY
TYPE_SCROLL_SENSITIVE	CONCUR_READ_ONLY
TYPE_SCROLL_SENSITIVE	CONCUR_UPDATABLE

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

Restricción: si el conjunto de resultados es desplazable y se utiliza para seleccionar columnas de una tabla en un servidor de DB2 Database para Linux, UNIX y Windows, la sentencia SELECT que sirve para definir el conjunto de resultados no puede incluir columnas que tengan los tipos de datos siguientes:

- BLOB
- CLOB
- XML
- Un tipo diferenciado que esté basado en cualquiera de los tipos de datos anteriores de esta lista

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.

Tabla 8. Métodos para posicionar un cursor desplazable en un conjunto de resultados

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

Notas:

1. Si el cursor está situado antes de la primera fila del conjunto de resultados, este método posiciona el cursor en la primera fila.
2. Si el cursor está situado después de la última fila del conjunto de resultados, este método posiciona el cursor en la última fila.
3. Si el valor absoluto de *n* es mayor que el número de filas del conjunto de resultados, este método posiciona el cursor después de la última fila si *n* es positivo, o antes de la primera fila si *n* es negativo.
4. El cursor debe estar en una fila válida del conjunto de resultados para poder utilizar este método. Si el cursor está antes de la primera fila o después de la última fila, el método emite una excepción de SQL.
5. Suponga que *m* es el número de filas del conjunto de resultados y *x* es la fila actual del conjunto de resultados. Si *n*>0 y *x*+*n*>*m*, el controlador posiciona el cursor 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 54 y necesita ver los valores más recientes de la fila actual, invoque el método refreshRow.

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

d. Ejecute una o más de las operaciones siguientes:

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

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

El valor *resultSetConcurrency* del conjunto de resultados debe ser CONCUR_UPDATABLE para 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.

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

```
String s;
Connection con;
Statement stmt;
ResultSet rs;
...
stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                           ResultSet.CONCUR_UPDATABLE);                    1
                                        // Crear un objeto Statement
                                        // para un Resultset desplazable
                                        // y actualizable
rs = stmt.executeQuery("SELECT EMPNO FROM EMPLOYEE FOR UPDATE OF PHONENO");
                                       // Crear el conjunto de resultados  3
rs.afterLast();                        // Posicionar el cursor al final
                                       // del conjunto de resultados       4a
while (rs.previous()) {                // Retroceder el cursor
 s = rs.getString("EMPNO");            // 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 16. Uso de un cursor desplazable

Comprobación de si la fila actual de un ResultSet es un hueco por supresión o 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 por supresión o actualización antes de intentar recuperar filas del ResultSet.

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:

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.

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 pueden 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.

Para insertar una fila en un ResultSet, siga estos pasos:

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 que aparece a continuación muestra la forma en que insertar una fila en un ResultSet que está formado por todas las filas en 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.

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

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 el ResultSet puede detectar inserciones, invoque el método ResultSet.rowInserted para determinar si se ha insertado la fila actual.

Operaciones de SQL sobre varias filas con el IBM Data Server Driver para JDBC y SQLJ

El 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

Cuando ejecuta un lote de sentencias INSERT y la fuente de datos es compatible con una operación INSERT de varias filas, el IBM Data Server Driver para JDBC y SQLJ utiliza la operación INSERT de varias filas para insertar las filas. La operación INSERT de varias filas puede proporcionar un mejor rendimiento que las sentencias INSERT individuales.

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

FETCH de varias filas

Cuando utiliza un cursor desplazable para realizar una operación FETCH en una aplicación JDBC, y el valor de la propiedad useRowsetCursor es `true` o no está definido, el IBM Data Server Driver para JDBC y SQLJ utiliza la operación FETCH de varias filas para recuperar las filas, si la fuente de datos es compatible con esa operación. La operación FETCH de varias filas puede proporcionar un mejor rendimiento que las sentencias FETCH individuales.

UPDATE o DELETE de posición de varias filas

El 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
```

Si utiliza la técnica JDBC 1 para actualizar o suprimir datos en una fuente de datos que es compatible con FETCH de varias filas, la sentencia UPDATE o DELETE de posición podría actualizar o suprimir varias filas cuando solamente se desea actualizar o suprimir una fila individual. Para evitar actualizaciones o supresiones inesperadas, puede realizar una de estas acciones:

- Utilice un ResultSet actualizable para recuperar y actualizar una sola fila cada vez.
- Utilice la cláusula FOR ROW *n* OF ROWSET en las sentencia UPDATE o DELETE para especificar una fila determinada que se debe modificar o suprimir.

Llamada a procedimientos almacenados en aplicaciones JDBC

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

Los pasos básicos para invocar procedimientos almacenados son:

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

   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. La tabla siguiente lista los tipos de literales permitidos, y sus correspondientes tipos de JDBC.

Tabla 9. 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
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'

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

   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.
3. Invoque el método CallableStatement.registerOutParameter para indicar qué parámetros son de solo salida (parámetros que se definen como OUT en la sentencia CREATE PROCEDURE) o cuáles son de entrada y salida (parámetros que se definen como INOUT en la sentencia CREATE PROCEDURE).

   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 salida, o de entrada y salida de la sentencia CALL **exactamente** tal 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.

   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 Dynamic Server (IDS) no son compatibles con la obtención de varios conjuntos de resultados.
5. Si el procedimiento almacenado devuelve varios conjuntos de resultados, recupere los conjuntos de resultados.

Restricción: Las fuentes de datos IDS no son compatibles con la obtención de varios 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: 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 situados a la derecha de determinadas sentencias corresponden a 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
```

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

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

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

Recuperación de un número conocido de conjuntos 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.

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

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: 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 conjunto de resultados
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.

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.

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 63 para obtener el conjunto de resultados siguiente.

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.

Para especificar si los conjuntos de resultados deben permanecer abiertos, siga este proceso:

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: 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
 ...                                        // Procesar 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
```

Datos LOB en aplicaciones JDBC con IBM Data Server Driver para JDBC y SQLJ

El 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 es compatible con la modalidad continua progresiva, el IBM Data Server Driver para JDBC y SQLJ puede utilizar esa modalidad para recuperar datos de 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 versiones posteriores es compatible con la modalidad continua progresiva para objetos 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.

Para el IBM Data Server Driver para JDBC y SQLJ Versión 3.50 y versiones posteriores, la modalidad continua progresiva es el comportamiento por omisión para recuperar objetos LOB en las conexiones con DB2 Database para Linux, UNIX y Windows Versión 9.5 y versiones posteriores.

La modalidad continua progresiva se puede utilizar en el IBM Data Server Driver para JDBC y SQLJ Versión 3.1 y versiones posteriores, pero para el IBM Data Server Driver para JDBC y SQLJ Versión 3.2 y versiones posteriores, 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 versiones posteriores.

Si el valor de la propiedad `progressiveStreaming` del IBM Data Server Driver para JDBC y SQLJ es `DB2BaseDataSource.YES`, significa que su aplicación utiliza la 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 el 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

El IBM Data Server Driver para JDBC y SQLJ puede utilizar localizadores de LOB para recuperar datos de columnas LOB. Debe utilizar localizadores de LOB solamente si la fuente de datos no es compatible con la modalidad continua progresiva.

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 localizadores progresivos:

- 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 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 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

El 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 el 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
  - getBytes

  Para columnas CLOB:
  - getAsciiStream
  - getCharacterStream
  - 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, el 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
  - setBinaryStream
  - setObject, donde el valor del parámetro *Object* es InputStream.

  Para columnas CLOB:
  - setString
  - setAsciiStream
  - setCharacterStream

– setObject, donde el valor del parámetro *Object* es Reader

Si especifica -1 para *length*, el 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 el IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 2, no puede invocar un procedimiento almacenado que tenga parámetros DBCLOB OUT o INOUT.

Si está utilizando el 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 el 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 el 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 de entrada de columnas BLOB, o parámetros de entrada/salida que se utilizan como entrada para columnas BLOB, puede utilizar uno de los métodos siguientes:

- Utilice una variable de entrada java.sql.Blob, que se corresponde exactamente con una columna BLOB:

  ```
  cstmt.setBlob(parmIndex, blobData);
  ```

- Utilice una llamada CallableStatement.setObject que especifique que el tipo de datos de destino es BLOB:

  ```
  byte[] byteData = {(byte)0x1a, (byte)0x2b, (byte)0x3c};
  cstmt.setObject(parmInd, byteData, java.sql.Types.BLOB);
  ```

- Utilice un parámetro de entrada de tipo java.io.ByteArrayInputStream con una llamada CallableStatement.setBinaryStream. Un objeto java.io.ByteArrayInputStream es compatible con un tipo de datos BLOB. Para esta llamada es necesario especificar la longitud exacta de los datos de entrada:

  ```
  java.io.ByteArrayInputStream byteStream =
    new java.io.ByteArrayInputStream(byteData);
  int numBytes = byteData.length;
  cstmt.setBinaryStream(parmIndex, byteStream, numBytes);
  ```

Parámetros de salida para columnas BLOB

Para los parámetros de salida de columnas BLOB, o parámetros de entrada/salida que se utilizan como salida para columnas BLOB, puede utilizar uno de los métodos siguientes:

- Utilice la llamada CallableStatement.registerOutParameter para especificar que un parámetro de salida es de tipo BLOB. Luego puede recuperar el valor del parámetro e insertarlo en cualquier variable cuyo tipo de datos sea compatible con un tipo de datos BLOB. Por ejemplo, el código siguiente permite recuperar un valor BLOB e insertarlo en una variable byte[]:

  ```
  cstmt.registerOutParameter(parmIndex, java.sql.Types.BLOB);
  cstmt.execute();
  byte[] byteData = cstmt.getBytes(parmIndex);
  ```

Parámetros de entrada para columnas CLOB

Para los parámetros de entrada de columnas CLOB, o parámetros de entrada/salida que se utilizan como entrada para columnas CLOB, puede utilizar uno de los métodos siguientes:

- Utilice una variable de entrada java.sql.Clob, que se corresponde exactamente con una columna CLOB:

  ```
  cstmt.setClob(parmIndex, clobData);
  ```

- Utilice una llamada CallableStatement.setObject que especifique que el tipo de datos de destino es CLOB:

  ```
  String charData = "CharacterString";
  cstmt.setObject(parmInd, charData, java.sql.Types.CLOB);
  ```

- Utilice uno de los tipos siguientes de parámetros de entrada:
  - Un parámetro de entrada java.io.StringReader con una llamada cstmt.setCharacterStream:

    ```
    java.io.StringReader reader = new java.io.StringReader(charData);
    cstmt.setCharacterStream(parmIndex, reader, charData.length);
    ```

  - Un parámetro java.io.ByteArrayInputStream con una llamada cstmt.setAsciiStream, para datos ASCII:

```
byte[] charDataBytes = charData.getBytes("US-ASCII");
java.io.ByteArrayInputStream byteStream =
  new java.io.ByteArrayInputStream (charDataBytes);
cstmt.setAsciiStream(parmIndex, byteStream, charDataBytes.length);
```

Para estas llamadas es necesario especificar la longitud exacta de los datos de entrada.

- Utilice un parámetro de entrada de tipo String con una llamada cstmt.setString:

  ```
  cstmt.setString(charData);
  ```

  Si la longitud de los datos es mayor que 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 de salida de columnas CLOB, o parámetros de entrada/salida que se utilizan como salida para columnas CLOB, puede utilizar uno de los métodos siguientes:

- Utilice la llamada CallableStatement.registerOutParameter para especificar que un parámetro de salida es de tipo CLOB. Luego puede recuperar el valor del parámetro e insertarlo en cualquier variable cuyo tipo de datos sea compatible con un tipo de datos CLOB. Por ejemplo, el código siguiente permite recuperar un valor CLOB e insertarlo en una variable de tipo String:

  ```
  cstmt.registerOutParameter(parmIndex, java.sql.Types.CLOB);
  cstmt.execute();
  String charData = cstmt.getString(parmIndex);
  ```
- Utilice la llamada CallableStatement.registerOutParameter para especificar que un parámetro de salida es de tipo VARCHAR o LONGVARCHAR:

  ```
  cstmt.registerOutParameter(parmIndex, java.sql.Types.VARCHAR);
  cstmt.execute();
  String charData = cstmt.getString(parmIndex);
  ```

  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, DB2 para i5/OS e IBM Informix Dynamic Server (IDS) son compatibles con el 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.

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, el 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, el 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 10. 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 i5/OS	ROWID_VALID_FOREVER
IDS	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: uso de PreparedStatement.setObject con un tipo de destino com.ibm.db2.jcc.DB2RowID: suponga que rwid es una instancia de com.ibm.db2.jcc.DB2RowID. Para establecer el parámetro 1, utilice este formato del método setObject:

```
ps.setObject (1, rwid);
```

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 17. Creación y utilización de un tipo diferenciado

Invocación de procedimientos almacenados con parámetros ARRAY en aplicaciones JDBC

Las aplicaciones JDBC que se ejecutan bajo el IBM Data Server Driver para JDBC y SQLJ pueden invocar procedimientos almacenados que tienen parámetros ARRAY.

Puede utilizar el método CallableStatement.setArray o CallableStatement.setObject para asignar un objeto java.sql.Array a un parámetro ARRAY de procedimiento almacenado.

Puede registrar un parámetro ARRAY de salida 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;
...
stmt = 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. Existen 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.

El 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.

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
Savepoint savept = con.setSavepoint();        // 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
```

Figura 18. Establecimiento, retrotracción y liberación de un punto de salvaguarda en una aplicación JDBC

Recuperación de claves de generación automática en aplicaciones JDBC

Mediante el 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 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 habilitar la recuperación de claves generadas automáticamente a partir de una tabla, es necesario que al insertar filas indique que deseará recuperar valores de claves generadas automáticamente. Para ello debe establecer un distintivo en una llamada de método Connection.prepareStatement, Statement.executeUpdate o Statement.execute. La sentencia que se ejecuta debe ser una sentencia INSERT o un INSERT dentro de una sentencia SELECT. De lo contrario, el controlador JDBC no tiene en cuenta el parámetro por el que se establece el distintivo.

Restricción: 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 un SQLException cuando el usuario invoca un método addBatch o executeBatch para un objeto PreparedStatement que está preparado para devolver claves generadas automáticamente.

Para recuperar claves generadas automáticamente a partir de una tabla utilizando métodos de JDBC 3.0, debe seguir estos pasos:

1. Utilice uno de los métodos siguientes para indicar que desea obtener claves generadas automáticamente:
   - Si piensa utilizar el método PreparedStatement.executeUpdate para insertar filas, invoque uno de estos formatos del método Connection.prepareStatement para crear un objeto PreparedStatement:

     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.

     ```
     Connection.prepareStatement(sentencia-sql,
       Statement.RETURN_GENERATED_KEYS);
     ```

     Los formatos siguientes solamente son válidos si la fuente de datos es compatible con columnas de identidad y con la sentencia INSERT dentro de una sentencia SELECT. *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 [] nombresColumnas);
     Connection.prepareStatement(sentencias-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.

     ```
     Statement.executeUpdate(sentencia_sql, Statement.RETURN_GENERATED_KEYS);
     ```

     Los formatos siguientes solamente son válidos si la fuente de datos es compatible con columnas de identidad y con la sentencia INSERT dentro de una sentencia SELECT. *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 [] nombresColumnas);
     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.

   El tipo de datos de las claves generadas automáticamente del objeto ResultSet es DECIMAL, con independencia del tipo de datos de la columna correspondiente.

El código siguiente crea una tabla con una columna de identidad, inserta 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.

Figura 19. Ejemplo de recuperación de claves generadas automáticamente

```
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 valores de claves
                                      // generadas 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.

Figura 20. Ejemplo de recuperación de claves generadas automáticamente después de ejecutar una sentencia INSERT de varias filas

```
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 valores de claves
                                        // generadas automáticamente
 System.out.println("valor de clave generada automáticamente = " + idColVar);
}
stmt.close();
con.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.

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, 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 11. 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
setDB2ClientUser	Nombre de usuario para una conexión
setDB2ClientWorkstation	Nombre de estación de trabajo del cliente para una conexión

Para definir la información ampliada sobre el cliente, siga estos pasos:

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 11.
4. Ejecute una sentencia de SQL para hacer que la información se envíe al servidor DB2.

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 que

aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
public class ClientInfoTest {
  public static void main(String[] args) {
    String url = "jdbc:db2://sysmvs1.stl.ibm.com:5021/san_jose";
    try {
      Class.forName("com.ibm.db2.jcc.DB2Driver");
      String user = "db2adm";
      String password = "db2adm";
      Connection 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 21. 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.

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:

1. Cree un objeto Connection.
2. Invoque el método java.sql.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.

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);
      conn.setClientInfo("ClientUser", "Michael L Thompson"); 2
      conn.setClientInfo("ClientWorkstation, "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 22. 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

Tabla 12. Valores de propiedades de información del cliente para DB2 Database para Linux, UNIX y Windows

NAME	MAX_LEN	DEFAULT_VALUE	DESCRIPTION
ApplicationName	255	Serie de caracteres vacía	Nombre de la aplicación que está actualmente haciendo uso de la conexión.
ClientAccountingInformation	255	Serie de caracteres vacía	Valor de la serie de caracteres contable de la información del cliente que está especificada para la conexión.
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.

Tabla 12. Valores de propiedades de información del cliente para DB2 Database para Linux, UNIX y Windows (continuación)

NAME	MAX_LEN	DEFAULT_VALUE	DESCRIPTION
ClientUser	255	Serie de caracteres vacía	Nombre del usuario para el que se ejecuta la aplicación que está haciendo uso de la conexión.

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

Tabla 13. Valores de propiedades de información del cliente para DB2 para z/OS

NAME	MAX_LEN	DEFAULT_VALUE para la conectividad de tipo 4	DEFAULT_VALUE para la conectividad de tipo 2	DESCRIPTION
ApplicationName	32	Valor de la propiedad clientProgramName, si está definida. ″db2jcc_application″ en otro caso.	Serie de caracteres vacía	Nombre de la aplicación que está actualmente haciendo uso de la conexión.
ClientAccountingInformation	200	Serie de caracteres que resulta de la concatenación de los valores siguientes: • ″JCC*nnnnn*″, donde *nnnnn* es el nivel del controlador, tal como 04000. • Valor establecido por DB2Connection.setDB2ClientWorkstation. Si el valor no está establecido, el valor por omisión es el nombre del sistema principal local. • Valor de la propiedad applicationName, si está definida. En otro caso, 20 espacios en blanco. • Valor de la propiedad clientUser, si está definida. En otro caso, ocho espacios en blanco.	Serie de caracteres vacía	Valor de la serie de caracteres contable de la información del cliente que está especificada para la conexión.
ClientHostname	18	Valor establecido por DB2Connection.setDB2ClientWorkstation. Si el valor no está establecido, el valor por omisión es el nombre del sistema principal local.	Serie de caracteres vacía	Nombre de sistema principal del sistema donde se ejecuta la aplicación que está haciendo uso de la conexión.
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.	Serie de caracteres vacía	Nombre del usuario para el que se ejecuta la aplicación que está haciendo uso de la conexión.

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.

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 distintivo de cambio de fila. El distintivo 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 horaria de cambio de fila, el distintivo de cambio de fila es totalmente exacto. Si crea la tabla sin una columna de indicación horaria de cambio de fila, o modifica una tabla para añadir una columna de indicación horaria de cambio de fila, el distintivo de cambio de fila puede que no refleje fielmente las actualizaciones hechas en una fila. Esto significa que el distintivo 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. Libere los bloqueos mantenidos en la tabla.
4. Actualice las filas seleccionadas, si el distintivo 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 distintivo 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 distintivo 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.
...
```

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 interno XML para almacenar datos XML en una columna en forma de conjunto estructurado de nodos en formato de árbol.

En las aplicaciones, los datos XML se encuentran en formato de serie serializada.

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

Cuando actualiza o inserta datos en columnas XML de una tabla de base de datos, los datos de entrada de sus aplicaciones JDBC deben tener el formato de serie de caracteres serializada.

La tabla siguiente lista los métodos y los correspondientes tipos de datos de entrada que puede utilizar para insertar datos en columnas XML.

Tabla 14. 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.setString	String

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"" xmlns=""http://posample.org"">...</customerinfo>";
  cid.setString(cidData);
                             // 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()
{
  PreparedStatement insertStmt = null;
  String sqls = null;
  int cid = 0;
  ResultSet rs=null;
  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: No record inserted.");
      }
    }
    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());

    }
  }
```

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.

Cuando recupera datos de columnas XML en una tabla DB2, los datos de salida tienen el formato de una serie de caracteres serializada. Esto ocurre independientemente de que se recupere todo el contenido de una columna XML o de una secuencia de la columna.

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.

  Los métodos SQLXML.getXXX añaden declaraciones XML con especificaciones de codificación a los datos de salida.
- 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.

  Los métodos DB2Xml.getDB2XmlXXX añaden declaraciones XML con especificaciones de codificación para los datos de salida. Los métodos DB2Xml.getDB2XXX no añaden declaraciones XML con especificaciones de codificación para los datos de salida.

  Esta técnica utiliza los objetos DB2Xml que están en desuso. Es preferible utilizar la técnica descrita anteriormente.

La tabla siguiente muestra los métodos ResultSet y sus correspondientes tipos de datos de salida para recuperar datos XML.

Tabla 15. Métodos ResultSet y tipos de datos para recuperar datos XML

Método	Tipo de datos de salida
ResultSet.getAsciiStream	InputStream

Tabla 15. Métodos ResultSet y tipos de datos para recuperar datos XML (continuación)

Método	Tipo de datos de salida
ResultSet.getBinaryStream	InputStream
ResultSet.getBytes	byte[]
ResultSet.getCharacterStream	Reader
ResultSet.getObject	DB2Xml
ResultSet.getSQLXML	SQLXML
ResultSet.getString	String

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 correspondientes tipos de datos de salida y el tipo de codificación utilizado en las declaraciones XML.

Tabla 16. 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	Ninguno
SQLXML.getString	String	Ninguno
DB2Xml.getDB2AsciiStream	InputStream	Ninguno
DB2Xml.getDB2BinaryStream	InputStream	Ninguno
DB2Xml.getDB2Bytes	byte[]	Ninguno
DB2Xml.getDB2CharacterStream	Reader	Ninguno
DB2Xml.getDB2String	String	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	String	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: 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()
  {
    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();
      String 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: en el ejemplo siguiente, se muestra la recuperación de datos desde una columna XML a una variable de tipo String.

```
public void fetchToString()
  {
    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();
      String 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()
  {
    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();
      String 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

Los procedimientos almacenados de SQL o externos y las función definida por el usuario externas pueden incluir parámetros XML.

Para los procedimientos de SQL, los parámetros de la definición de procedimiento almacenado tienen el tipo XML. Para los procedimientos almacenados externos y las función definida por el usuario, 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 ocom.ibm.db2.jcc.DB2Xml. Para registrar parámetros de salida XML, registre los parámetros como pertenecientes al tipo java.sql.Types.SQLXML ocom.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 necesita tener instalado JDBC 4.0.

```
java.sql.SQLXML in_xml = xmlvar;
java.sql.SQLXML out_xml = null;
java.sql.SQLXML inout_xml = xmlvar;
                                  // Declarar un parámetro XML de entrada,
                                  // de salida y de entrada/salida
Connection con;
CallableStatement cstmt;
ResultSet rs;
...
stmt = con.prepareCall("CALL SP_xml(?,?,?)");
                                  // Crear un objeto CallableStatement
cstmt.setObject (1, in_xml);      // Definir parámetro de entrada
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
System.out.println("Valores de parámetros de llamada a SP_xml: ");
System.out.println("Valor de parámetro de salida ");
printString(out_xml.getString());
                           // Utilizar el método getBytes de SQLXML.getString
                           // para convertir el valor en una serie de
                           // caracteres para su impresión
System.out.println("Valor de parámetro de entrada/salida ");
printString(inout_xml.getString());
```

Para invocar una rutina con parámetros XML desde un programa SQLJ, utilice parámetros del tipo `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 necesita tener instalado JDBC 4.0.

```
java.sql.SQLXML in_xml = xmlvar;
java.sql.SQLXML out_xml = null;
java.sql.SQLXML inout_xml = xmlvar;
                                  // Declarar un parámetro XML de entrada,
                                  // de salida y de entrada/salida
...
#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 ");
printString(out_xml.getString());
                           // Utilizar el método SQLXML.getString
                           // para convertir el valor en una serie
                           // de caracteres para su impresión
System.out.println("Valor de parámetro de entrada/salida ");
printString(inout_xml.getString());
```

Soporte de Java para el registro y la eliminación de esquemas XML

El 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.

Estos métodos son equivalentes a los procedimientos almacenados SYSPROC.XSR_REGISTER, SYSPROC.XSR_ADDSCHEMADOC,

SYSPROC.XSR_COMPLETE, SYSPROC.XSR_REMOVE y SYSPROC.XSR_UPDATE proporcionados por DB2. Los métodos JDBC 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, elimina el esquema XML cuyo contenido se copia.

Antes de poder invocar estos métodos, los procedimientos almacenados subyacentes se deben instalar 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. El valor *xmlSchemaLocations* es nulo, por lo que DB2 no encontrará este esquema XML en una invocación de DSN_XMLVALIDATE que proporcione un valor no nulo para la ubicación del esquema XML. No se registran propiedades adicionales.

Figura 23. Ejemplo de registro de un esquema XML en DB2 utilizando un documento XML procedente de una corriente de datos de entrada.

```
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.

Figura 24. Ejemplo de eliminación de un esquema XML en DB2

```
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 muestra 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.

Figura 25. Ejemplo de actualización de un esquema XML

```
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 *level* que puede especificar en el método Connection.setTransactionIsolation y sus equivalentes para el servidor de bases de datos.

Tabla 17. Niveles de aislamiento equivalentes para JDBC y DB2

Valor para JDBC	Nivel de aislamiento de DB2
TRANSACTION_SERIALIZABLE	Lectura repetible
TRANSACTION_REPEATABLE_READ	Estabilidad de lectura
TRANSACTION_READ_COMMITTED	Estabilidad del cursor
TRANSACTION_READ_UNCOMMITTED	Lectura no confirmada

Confirmación o retrotracción de transacciones JDBC

En JDBC, para confirmar o retrotraer explícitamente transacciones, utilice los métodos commit o rollback.

Por ejemplo:

```
 Connection con;
 ...
con.commit();
```

Si la modalidad de confirmación automática (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 IDS

Para las conexiones con fuentes de datos IDS, la modalidad de confirmación automática por omisión depende del tipo de la fuente de datos. La tabla siguiente muestra los valores por omisión.

Tabla 18. Modalidades de confirmación automática para fuentes de datos IDS

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 registro cronológico	false	No aplicable
Base de datos no compatible con ANSI con registro cronológico	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.

El 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 el 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 del 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

El 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, el 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, el 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 el 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.

Estos son los pasos básicos para el manejo de una excepción SQLException en un programa JDBC que se ejecuta con el IBM Data Server Driver para JDBC y SQLJ:

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 una fuente de datos DB2 para z/OS o IBM Informix Dynamic Server (IDS), establezca la propiedad setRetrieveMessagesFromServerOnGetMessage en `true` si desea obtener el texto de mensaje completo al invocar SQLException.getMessage.
3. Coloque código que pueda generar una excepción SQLException en un bloque `try`.
4. 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 IDS, 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. 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) Transforme 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.
d. Invoque el método SQLException.getNextException para recuperar la excepción SQLException siguiente.

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 26. Proceso de una excepción de SQL cuando se utiliza el 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;";
                                  // Definir propiedades para recuperar
                                  // texto de mensaje completo
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 {                                                                    3
  // Código que podría generar excepciones de SQL
 ...
} catch(SQLException sqle) {
    while(sqle != null) {             // Comprobar si existen más       4a
                                      // excepciones de SQL para procesar
 //=====> Proceso opcional de errores específicos de
 //  IBM Data Server Driver para JDBC y SQLJ
      if (sqle instanceof DB2Diagnosable) {                              4c
      // 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;                        4c1
          diagnosable.printTrace (printWriter, "");                      4c2
          java.lang.Throwable throwable =
            diagnosable.getThrowable();                                  4c3
          if (throwable != null) {
            // Extraer información sobre java.lang.Throwable,
            // tal como mensaje o rastreo de pila.
            ...
          }
          DB2Sqlca sqlca = diagnosable.getSqlca();                       4c4
                                        // Obtener objeto DB2Sqlca
          if (sqlca != null) {            // Comprobar que DB2Sqlca no es nulo
            int sqlCode = sqlca.getSqlCode(); // Obtener cód. error SQL  4c5
            String sqlErrmc = sqlca.getSqlErrmc();                       4c6
                                        // Obtener el valor SQLERRMC completo
            String[] sqlErrmcTokens = sqlca.getSqlErrmcTokens();
                                        // También se pueden recuperar
                                        // distintivos SQLERRMC individuales
            String sqlErrp = sqlca.getSqlErrp();                         4c7
                                        // Obtener el valor SQLERRP
            int[] sqlErrd = sqlca.getSqlErrd();                          4c8
                                        // Obtener campos SQLERRD
            char[] sqlWarn = sqlca.getSqlWarn();                         4c9
                                        // Obtener campos SQLWARN
            String sqlState = sqlca.getSqlState();                       4c10
                                        // Obtener SQLSTATE
            String errMessage = sqlca.getMessage();                      4c11
                                        // 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(); // Recup. excepción SQL siguiente    4d
        }
    }
```

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.

Estos son los pasos básicos para recuperar información de aviso de SQL:

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 procedente de una fuente de datos DB2 para z/OS o IBM Informix Dynamic Server (IDS) al ejecutar llamadas a SQLWarning.getMessage, establezca la propiedad setRetrieveMessagesFromServerOnGetMessage en `true`.
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. Si no lo es, 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.

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;";
                                  // Definir propiedades para recuperar
                                  // texto de mensaje completo
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 27. 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.

Para recuperar información de la excepción BatchUpdateException, siga estos pasos:

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.

   Statement.EXECUTE_FAILED
   Este valor se devuelve si la sentencia no se ha ejecutado satisfactoriamente.
2. Utilice los métodos getMessage, getSQLState y getErrorCode de SQLException para obtener la descripción del error, el estado de SQL y el código de error correspondientes al primer error.
3. Utilice el método BatchUpdateException.getNextException para obtener una excepción SQLException encadenada.
4. Ejecute en bucle las llamadas de método getMessage, getSQLState, getErrorCode y getNextException para obtener información sobre una excepción SQLException y obtener la siguiente excepción SQLException.

El siguiente fragmento de código de programa muestra cómo obtener los campos de una excepción BatchUpdateException y los objetos encadenados SQLException. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
try {
  // Actualizaciones por lotes
} catch(BatchUpdateException buex) {
    System.err.println("Contents of BatchUpdateException:");
    System.err.println(" Update counts: ");
    int [] updateCounts = buex.getUpdateCounts();            1
    for (int i = 0; i < updateCounts.length; i++) {
      System.err.println("  Statement " + i + ":" + updateCounts[i]);
    }
    System.err.println(" Message: " + buex.getMessage());    2
    System.err.println(" SQLSTATE: " + buex.getSQLState());
    System.err.println(" Error code: " + buex.getErrorCode());
    SQLException ex = buex.getNextException();               3
    while (ex != null) {                                     4
      System.err.println("SQL exception:");
      System.err.println(" Message: " + ex.getMessage());
      System.err.println(" SQLSTATE: " + ex.getSQLState());
      System.err.println(" Error code: " + ex.getErrorCode());
      ex = ex.getNextException();
    }
}
```

Figura 28. Recuperación de los campos de una excepción BatchUpdateException

Manejo de una excepción de SQL cuando se utiliza el controlador JDBC de DB2 de tipo 2

Al igual que ocurre en todos los programas Java, el manejo de errores cuando se utiliza el controlador JDBC de DB2 de tipo 2 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.

JDBC proporciona la clase SQLException para manejar errores. Todos los métodos de JDBC emiten una instancia de SQLException cuando se produce un error durante la ejecución del método. De acuerdo con la especificación JDBC, un objeto SQLException contiene la información siguiente:

- Un objeto `String` que contiene una descripción del error o el valor nulo
- Un objeto `String` que contiene el SQLSTATE o el valor nulo
- Un valor `int` que contiene un código de error
- Un puntero que indica la ubicación del objeto SQLException siguiente o un valor nulo

Estos son los pasos básicos para manejar un SQLException en un programa JDBC que se ejecuta bajo el controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows (controlador JDBC de DB2 de tipo 2):

1. Coloque código que pueda generar una excepción SQLException en un bloque `try`.
2. En el bloque `catch`, ejecute los pasos siguientes en un bucle:
   a. Determine si ha recuperado la última excepción SQLException. En caso negativo, continúe en el paso siguiente.
   b. Recupere la información de error procedente de SQLException.
   c. Invoque el método SQLException.getNextException para recuperar la excepción SQLException siguiente.

El código de programa siguiente muestra un bloque `catch` que hace uso de la versión de SQLException para DB2 que se proporciona con el controlador JDBC de DB2 de tipo 2. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
import java.sql.*;           // Importar paquete de la API de JDBC
...
try {
  // Código de programa que podría generar excepciones de SQL                  1
 ...
} catch(SQLException sqle) {
    while(sqle != null) {             // Comprobar si existen más             2a
      System.out.println("Message:  " + sqle.getMessage());                    2b
      System.out.println("SQLSTATE: " + sqle.getSQLState());
      System.out.println("Código de error SQL: " + sqle.getErrorCode());
      sqle=sqle.getNextException();     // Recuperar excepción de SQL siguiente 2c
    }
}
```

Figura 29. Proceso de una excepción SQLException cuando se utiliza el IBM Data Server Driver para JDBC y SQLJ

Manejo de un aviso de SQL con el controlador JDBC de DB2 de tipo 2

A diferencia de los errores de SQL, los avisos de SQL no hacen que los métodos de JDBC emitan excepciones. En lugar de ello, las clases Connection, Statement, PreparedStatement, `CallableStatement` y ResultSet contienen métodos getWarnings, que es necesario invocar después de ejecutar sentencias de SQL para determinar si se han producido avisos de SQL.

La invocación de getWarnings hace que se recupere un objeto SQLWarning.

El controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows (controlador JDBC de DB2 de tipo 2) genera objetos SQLWarning genéricos. Un objeto SQLWarning genérico contiene la información siguiente:

- Un objeto `String` que contiene una descripción del aviso o el valor nulo
- Un objeto `String` que contiene el SQLSTATE o el valor nulo
- Un valor `int` que contiene un código de error
- Un puntero que indica la ubicación del objeto SQLWarning siguiente o un valor nulo

Estos son los pasos básicos para recuperar información de aviso de SQL:

1. Inmediatamente después de invocar un método por el que se ejecuta una sentencia de SQL, invoque el método getWarnings para obtener un objeto SQLWarning.
2. Ejecute en bucle los pasos siguientes:
   a. Determine si el objeto SQLWarning es nulo. Si no lo es, continúe en el paso siguiente.
   b. Invoque el método SQLWarning.getMessage para obtener la descripción del aviso.
   c. Invoque el método SQLWarning.getSQLState para obtener el valor SQLSTATE.
   d. Invoque el método SQLWarning.getErrorCode para obtener el valor del código de error.
   e. Invoque el método SQLWarning.getNextWarning para recuperar el objeto SQLWarning siguiente.

El código siguiente muestra cómo obtener información genérica sobre SQLWarning. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
Connection con;
Statement stmt;
ResultSet rs;
SQLWarning sqlwarn;
...
stmt = con.createStatement();         // Crear un objeto Statement
rs = stmt.executeQuery("SELECT * FROM EMPLOYEE");
                                      // Obtener tabla de resultados de la consulta
sqlwarn = stmt.getWarnings();         // Obtener avisos producidos            1
while (sqlwarn != null) {             // Mientras haya avisos, obtener e      2a
                                      // imprimir información de aviso
  System.out.println ("Descripción del aviso: " + sqlwarn.getMessage());  2b
  System.out.println ("SQLSTATE: " + sqlwarn.getSQLState());              2c
  System.out.println ("Código de error: " + sqlwarn.getErrorCode());      2d
  sqlwarn=sqlwarn.getNextWarning();    // Obtener aviso de SQL siguiente  2e
}
```

Figura 30. Proceso de un aviso de SQL

Soporte de redireccionamiento de cliente de IBM Data Server Driver para JDBC y SQLJ

La función de redireccionamiento de cliente automático de DB2 permite a las aplicaciones de cliente recuperarse de una pérdida de comunicación con el servidor con el fin de que puedan continuar trabajando con el mínimo de interrupciones posible. Las aplicaciones cliente de JDBC y SQLJ pueden sacar partido de dicho soporte.

Cuando se bloquea un servidor, cada cliente que está conectado a dicho servidor recibe un error de comunicación que finaliza la conexión y provoca un error de la aplicación. Si la disponibilidad es importante, deberá implementar una configuración redundante o contar con soporte para la función de gestión de anomalías. La gestión de anomalías es la capacidad de un servidor para asumir operaciones cuando falla otro servidor. En cualquiera de los dos casos, el cliente de IBM Data Server Driver para JDBC y SQLJ intenta restablecer la conexión con el servidor original o con un servidor nuevo. Una vez restablecida la conexión, la aplicación recibirá una excepción SQLException que le notificará el error de la transacción; sin embargo, la aplicación podrá seguir con la transacción siguiente.

El soporte de redireccionamiento del cliente IBM Data Server Driver para JDBC y SQLJ está disponible para IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 4 para servidores DB2 Database para Linux, UNIX y Windows o DB2 para z/OS. El redireccionamiento del cliente es efectivo para las conexiones obtenidas utilizando la interfaz javax.sql.DataSource, javax.sql.ConnectionPoolDataSource, `javax.sql.XADataSource` o java.sql.DriverManager.

Para definir los servidores primario y alternativo para el redireccionamiento del cliente cuando utiliza la interfaz DataSource o `ConnectionPoolDataSource`, defina la propiedad clientRerouteServerListJNDIName.

Para definir los servidores primario y alternativo para el redireccionamiento del cliente cuando utiliza la interfaz DriverManager, defina las propiedades clientRerouteAlternateServerName y clientRerouteAlternatePortNumber.

Si no define propiedades del controlador JDBC para los servidores primario y alternativo, se puede producir el redireccionamiento del cliente si las direcciones de los servidores primario y alternativo están definidas en entradas de DNS. El IBM Data Server Driver para JDBC y SQLJ busca información sobre los servidores primario y alternativo en las entradas de DNS si esa información no está contenida en la memoria del controlador JDBC y JDNI no se utiliza.

Restricción: La utilización del redireccionamiento del cliente para conexiones establecidas mediante la interfaz DriverManager está sujeta a 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 del cliente no está habilitado para las conexiones por omisión (jdbc:default:connection).

Configuración de los servidores DB2 para z/OS para el redireccionamiento de cliente

Si el direccionamiento de Sysplex está inhabilitado en un servidor DB2 para z/OS, y está configurado un grupo de compartimiento de datos para el acceso a un miembro determinado, puede utilizar el soporte de redireccionamiento del cliente del IBM Data Server Driver para JDBC y SQLJ.

Si se habilita el direccionamiento de Sysplex para un grupo de compartimiento de datos de DB2, el trabajo se distribuye entre todos los miembros de un grupo de compartimiento de datos por medio del servidor de DB2 para z/OS y del Gestor de carga de trabajo para z/OS (WLM). Si falla uno de los miembros de un grupo de compartimiento de datos de DB2, el trabajo se transfiere automáticamente a otros miembros. En este caso, no es necesario el soporte de redireccionamiento del cliente del IBM Data Server Driver para JDBC y SQLJ.

La configuración del acceso específico de miembros incluye la configuración de un alias de ubicación que representa uno o más miembros del grupo de compartimiento de datos. Antes de que una aplicación de cliente de JDBC o SQLJ pueda utilizar el recurso de redireccionamiento de cliente, ha de existir un alias de ubicación que represente al menos dos miembros del grupo de compartimiento de datos. La aplicación cliente se conecta a dichos miembros utilizando el alias de ubicación, en vez de utilizar el nombre de ubicación para conectarse a todo el grupo. Si falla un miembro, se intentan las conexiones con otros miembros del grupo de compartimiento de datos, basándose en su prioridad (información del peso de servidor de WLM).

El soporte de redireccionamiento del cliente del IBM Data Server Driver para JDBC y SQLJ da soporte a los trabajos únicamente para el acceso específico de miembros. No es efectivo para el acceso a grupos. Por tanto, únicamente debería definirse una dirección de IP virtual dinámica de miembro (DVIPA). No debería definirse un grupo DVIPA.

Para configurar el grupo de compartimiento de datos para que las aplicaciones cliente de JDBC y SQLJ puedan sacar partido del soporte de redireccionamiento de cliente, el administrador de base de datos ha de realizar estos pasos:

1. Especifique el número de puerto de DRDA en el que todos los miembros estén a la escucha de las solicitudes de SQL.
2. Especifique un número de puerto de resincronización exclusiva para cada miembro.
3. Designe subconjuntos de miembros a los que pueden conectarse los solicitantes de DRDA.
4. Especifique un nombre de LU genérica para el grupo de compartimiento de datos, si se utiliza RACF PassTickets.

Consulte el tema "Configuración de grupos de compartimiento de datos como servidores TCP/IP" en el Centro de información de soluciones del Software de gestión de información para z/OS en la dirección http://publib.boulder.ibm.com/infocenter/imzic para obtener más detalles.

Operación de redireccionamiento del cliente de IBM Data Server Driver para JDBC y SQLJ sobre el cliente

El redireccionamiento del cliente del IBM Data Server Driver para JDBC y SQLJ es efectivo para las conexiones obtenidas mediante la interfaz javax.sql.DataSource, javax.sql.ConnectionPoolDataSource, javax.sql.XADataSource o java.sql.DriverManager.

El redireccionamiento del cliente en el cliente opera de esta manera:

1. Antes de establecer la primera conexión con la fuente de datos, el IBM Data Server Driver para JDBC y SQLJ obtiene información sobre el servidor primario y el servidor alternativo.
   a. Si las propiedades clientRerouteAlternateServerName y clientRerouteAlternatePortNumber DataSource están establecidas, el IBM Data Server Driver para JDBC y SQLJ carga esos valores en la memoria como valores del servidor alternativo, junto con los valores serverName y portNumber del servidor primario.
   b. Si las propiedades clientRerouteAlternateServerName y clientRerouteAlternatePortNumber DataSource no están establecidas, y está configurado un almacén JNDI mediante la propiedad clientRerouteServerListJNDIName en DB2BaseDataSource, el IBM Data Server Driver para JDBC y SQLJ carga en la memoria la información del servidor primario y servidor alternativo contenida en el almacén JNDI.
   c. Si no hay ninguna propiedad DataSource establecida para los servidores alternativos, y JNDI no está configurado, el IBM Data Server Driver para JDBC y SQLJ examina tablas DNS para encontrar información del servidor primario y alternativo. Si existe información de DNS, el IBM Data Server Driver para JDBC y SQLJ carga esos valores en la memoria.
   d. Si no está disponible ninguna información del servidor primario o alternativo, no se puede establecer una conexión y el IBM Data Server Driver para JDBC y SQLJ emite una excepción.
2. El IBM Data Server Driver para JDBC y SQLJ conecta con la fuente de datos utilizando el nombre y número de puerto del servidor primario.
3. Si falla la conexión con el servidor primario:
   a. El IBM Data Server Driver para JDBC y SQLJ intenta reconectar con el servidor primario.
   b. Si falla la reconexión con el servidor primario, el IBM Data Server Driver para JDBC y SQLJ intenta conectar con los servidores alternativos.

La reconexión con el servidor primario se denomina *conexión de recuperación*. La conexión con un servidor alternativo se denomina *conexión de relevo*.

El IBM Data Server Driver para JDBC y SQLJ utiliza las propiedades maxRetriesForClientReroute y retryIntervalForClientReroute para determinar cuántas veces reintentar la conexión y cuánto 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 de relevo es satisfactoria durante la conexión inicial, el controlador genera un SQLWarning. Si se produce una conexión de relevo satisfactoria después de la conexión inicial, el controlador emite una SQLException a la aplicación y devuelve el código de error -4498 para indicar a la aplicación que la conexión con el servidor alternativo se ha restablecido automáticamente y la transacción se ha retrotraído implícitamente. La aplicación puede entonces intentar su transacción sin realizar primero una retrotracción explícita.

   Después de una conexión satisfactoria con servidores alternativos, Connection o DataSource tiene las propiedades originales, excepto las referentes al nombre de servidor y número de puerto. Además, los registros especiales de la fuente de datos que se modificaron durante la conexión original se restablecen en la conexión de relevo.

   Puede invocar el método DB2Connection.alternateWasUsedOnConnect para determinar si se utilizó información de servidores alternativos al establecer una conexión. Si la fuente de datos devuelve información sobre el servidor primario y alternativo que es diferente de la lista contenida en la memoria, se actualiza la información sobre el servidor primario y alternativo contenida en la memoria. El valor devuelto por DB2Connection.alternateWasUsedOnConnect depende de la información más reciente sobre servidores alternativos contenida en la memoria, y no de la configuración inicial de DB2BaseDataSource.

Soporte de redireccionamiento del cliente del IBM Data Server Driver para JDBC y SQLJ con JNDI

Si crea conexiones mediante la interfaz javax.sql.DataSource, javax.sql.ConnectionPoolDataSource o javax.sql.XADataSource, puede indicar al IBM Data Server Driver para JDBC y SQLJ que JNDI está configurado para el redireccionamiento del cliente; para ello, defina la propiedad clientRerouteServerListJNDIName.

Puede también especificar la propiedad clientRerouteServerListJNDIContext, que proporciona el contexto JNDI utilizado para la vinculación y búsqueda de una instancia de DB2ClientRerouteServerList.

Para el redireccionamiento del cliente con JNDI, la información sobre el servidor primario y alternativo contenida en la memoria es una instancia de la clase DB2ClientRerouteServerList, que implementa la interfaz javax.naming.Referenceable.

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[]

Nombre de propiedad	Tipo de datos
com.ibm.db2.jcc.DB2ClientRerouteServerList.primaryPortNumber	int[]

Para cada propiedad se definen los métodos getXXX y setXXX.

Cuando una fuente de datos DataSource se configura para utilizar JNDI a fin de almacenar información alternativa sobre el redireccionamiento 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, el IBM Data Server Driver para JDBC y SQLJ intenta crear un 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 un relevo de funciones tras un error:

- El 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.

Para configurar el almacenamiento de modo que DB2ClientRerouteServerList sea permanente, siga los pasos siguientes:

1. Cree una instancia de DB2ClientRerouteServerList y vincule esa instancia con el registro de JNDI.

   Ejemplo:

   ```
   // 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("mvs1.sj.ibm.com");

   // Definir el número de puerto y nombre de servidor para el servidor alternativo
   int[] port = {50002};
   String[] server = {"mvs3.sj.ibm.com"};
   address.setAlternatePortNumber(port);
   address.setAlternateServerName(server);

   registry.rebind("serverList", address);
   ```

2. Asigne el nombre de JNDI del objeto DB2ClientRerouteServerList a la propiedad clientRerouteServerListJNDIName.

   Ejemplo:

   ```
   datasource.setClientRerouteServerListJNDIName("serverList");
   ```

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.

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 Dynamic Server, si la base de datos es compatible con el registro cronológico 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 31. 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");
      }
      catch( SQLException e )                                       5
      {
         System.out.println ("**** Excepción de SQL de SELECT...");
         while(e!=null) {
           System.out.println ("Mensaje de error: " + e.getMessage());
           System.out.println ("SQLSTATE: " + e.getSQLState());
           System.out.println ("Código de error: " + e.getErrorCode());
           e = e.getNextException(); // Buscar excepciones encadenadas
         }
      }
      catch (Exception e)
      {
         System.out.println("**** Excepción no de SQL  = " + e);
         e.printStackTrace();
      }
      try
      {
        #sql [ctx]                                                  4d
          {UPDATE DEPARTMENT SET MGRNO=:hvmgr
             WHERE DEPTNO=:hvdeptno};   // Actualizar datos para un departamento
                                                                    6
        #sql [ctx] {COMMIT}; // Confirmar actualización
      }
      catch (SQLException e)
      {
         System.out.println ("**** Excepción de SQL de UPDATE...");
         System.out.println ("Msje de error: " + e.getMessage() + ".  SQLSTATE=" +
           e.getSQLState() + "Código de error=" + e.getErrorCode());
         e.printStackTrace();
      }
      catch (Exception e)
      {
         System.out.println("**** Excepción no de SQL  = " + e);
         e.printStackTrace();
      }
      iter.close();        // Cerrar el iterador
      ctx.close();                                                  7
    }
    catch (SQLException e)
    {
       System.out.println ("**** Excepción de SQL ...");
       System.out.println ("Msje de error: " + e.getMessage() + ".  SQLSTATE=" +
              e.getSQLState() + "Código de error=" + e.getErrorCode());
       e.printStackTrace();
    }
    catch (Exception e)
    {
      System.out.println("**** Excepción no de SQL  = " + e);
      e.printStackTrace();
    }

  }
```

Notas para la Figura 31 en la página 111:

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 (cargar el controlador JDBC) no es necesario si utiliza JDBC 4.0.
4a, 4b, 4c y 4d	Estas sentencias muestran cómo ejecutar sentencias de SQL en SQLJ. La sentencia 4a es el equivalente de SQLJ para declarar un cursor de SQL. Las sentencias 4b y 4c son un equivalente de SQLJ para ejecutar sentencias FETCH de SQL. La sentencia 4d es el equivalente de SQLJ para ejecutar una sentencia UPDATE de SQL. Para obtener más información, consulte ″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.

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 una conexión creada 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.

Para utilizar la técnica de conexión 1 de SQLJ, siga estos pasos:

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 el 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.
   - Para el controlador JDBC de DB2 de tipo 2, invoque Class.forName de esta manera:

     ```
     Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");
     ```
3. Invoque el constructor para la clase de contexto de conexión que creó en el paso 1 en la página 113.

   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 cadena 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 el controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows (controlador JDBC de DB2 de tipo 2), debe especificar solamente las propiedades `user` y `password`. Para el 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.

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 ejecutar una sentencia SQL
```

Figura 32. 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.

Para utilizar la técnica de conexión 2 de SQLJ, siga estos pasos:

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 el 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.
   - Para el controlador JDBC de DB2 de tipo 2, invoque Class.forName de esta manera:

     ```
     Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");
     ```
3. Invoque el 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 cadena 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 115

   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 115.

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 ejecutar una sentencia de SQL
```

Figura 33. 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.

Para utilizar la técnica de conexión 3 de SQLJ, siga estos pasos:

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 116.

   Esto crea un objeto de contexto de conexión que especificará en cada sentencia de SQL que ejecute en la fuente de datos asociada. La sentencia de invocación del constructor debe tener el formato siguiente:

   ```
   clase-contexto-conexión objeto-contexto-conexión=
     new clase-contexto-conexión(Connection objeto-conexión-JDBC);
   ```

   El parámetro *objeto_conexión_JDBC* es el objeto Connection que creó en el paso 3.

El código 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 situados a la derecha de determinadas sentencias corresponden a 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 ejecutar una sentencia de SQL
```

Figura 34. 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.

Para utilizar la técnica de conexión 4 de SQLJ, siga estos pasos:

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.

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 ejecutar una sentencia de SQL
```

Figura 35. Uso de la técnica de conexión 4 para conectar con una fuente de datos

Técnica de conexión 5 de SQLJ: Utilización de una conexión creada previamente

La técnica de conexión 5 de SQLJ utiliza una conexión creada previamente para conectar con la fuente de datos.

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.

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 ejecutar una 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.

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.

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`. El valor `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 cadenas de caracteres con significados especiales en los programas SQLJ. Tenga en cuenta las restricciones siguientes cuando utilice esas cadenas de caracteres en sus programas SQLJ:

- La cadena __sJT_ es un prefijo reservado que se utiliza para los nombres de variables generados por SQLJ. No comience los siguientes tipos de nombres con __sJT_:
  - Nombres de expresiones de lenguaje principal
  - Nombres de variables Java 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 cadena _SJ es un sufijo reservado que se utiliza para archivos de recursos y clases que han sido creados por SQLJ. Evite utilizar la cadena _SJ en nombres de clases y nombres de archivos fuente de entrada.

Comentarios en una aplicación SQLJ

Para documentar su programa SQLJ, es necesario que incluya comentarios. Para ello, utilice comentarios de Java. Los comentarios Java están indicados mediante /* */ o //.

Puede incluir comentarios Java fuera de las cláusulas SQLJ allí donde el lenguaje Java lo permita. Dentro de una cláusula SQLJ, puede utilizar comentarios Java en los lugares siguientes:

- Dentro de una expresión de lenguaje principal (/* */ o //).
- Dentro de una cláusula ejecutable de una sentencia de SQL, si la fuente de datos permite la utilización de comentarios dentro de la sentencia de SQL (/* */ o --).

  Los pares /* y */ pueden estar anidados dentro de una sentencia de SQL.

Ejecución de sentencias de SQL en aplicaciones SQLJ

En un programa SQL normal, puede ejecutar sentencias de SQL para crear tablas, insertar, actualizar, suprimir o fusionar 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 DB2 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.

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.

Los pasos básicos son:

1. Declare el iterador.

   El iterador puede ser de posición o de nombre. Para las operaciones UPDATE o DELETE de posición, el iterador se debe declarar como actualizable. Para ello, la declaración debe incluir las cláusulas siguientes:

   implements sqlj.runtime.ForUpdate
   Esta cláusula hace que la clase de iterador creada incluya métodos para utilizar iteradores actualizables. Esta cláusula es necesaria para los programas con operaciones UPDATE o DELETE de posición.

   with (updateColumns=*"lista-columnas"***)**
   Esta cláusula especifica una lista de las columnas, separadas por comas, de la tabla de resultados que serán actualizadas por el iterador. Esta cláusula es opcional.

   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.

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 36. 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 enum;       // Declarar variable de lenguaje principal para contener
  BigDecimal sal;    // los valores de las columnas EMPNO y SALARY
  #sql [myConnCtx]
    upditer = {SELECT EMPNO, SALARY FROM EMPLOYEE                          4
      WHERE WORKDEPT='D11'};
                     // Asignar tabla de resultados a objeto de iterador
  #sql {FETCH :upditer INTO :enum,:sal};                                   5a
                           // Avanzar cursor hasta la fila siguiente
  while (!upditer.endFetch())                                              5b
                           // Determinar si iterador apunta a una fila
  {
    #sql [myConnCtx] {UPDATE EMPLOYEE SET SALARY=SALARY*1.05
      WHERE CURRENT OF :upditer};                                          5c
                           // Ejecutar actualización de posición
    System.out.println("Actualizando fila para " + enum);
    #sql {FETCH :upditer INTO :enum,:sal};
                           // Avanzar cursor hasta la fila siguiente
  }
  upditer.close();         // Cerrar el iterador                          6
  #sql [myConnCtx] {COMMIT};
                           // Confirmar los cambios
  myConnCtx.close();       // Cerrar el contexto de conexión
}
```

Figura 37. 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 38. 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 enum;               // Declarar variable lenguaje principal para
                             // recibir valores de la columnas 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
  {
    enum = upditer.EmpNo();  // Obtener número de empleado en la fila actual
    #sql [myConnCtx]
      {UPDATE EMPLOYEE SET SALARY=SALARY*1.05
        WHERE CURRENT OF :upditer};                                         5c
                             // Ejecutar actualización de posición
    System.out.println("Actualizando fila para " + enum);
  }
  upditer.close();           // Cerrar el iterador                         6
  #sql [myConnCtx] {COMMIT};
                             // Confirmar los cambios
  myConnCtx.close();         // Cerrar el contexto de conexión
}
```

Figura 39. 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 40. UPDATE de posición estática que ejecuta incorrectamente

En este ejemplo sólo se define un iterador. Se definen 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 40 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 41 en la página 128 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 41. 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

El 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.

Puede incluir los tipos siguientes de sentencias en una actualización por lotes:
- Sentencias INSERT, UPDATE y DELETE de búsqueda
- Sentencias CREATE, ALTER, DROP, GRANT y REVOKE
- Sentencias CALL con parámetros de entrada solamente

A diferencia de JDBC, SQLJ permite utilizar lotes heterogéneos que contienen sentencias con parámetros de entrada o expresiones de lenguaje principal. 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

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. Consulte ″Recuperación de información a partir de una excepción BatchUpdateException″ para obtener información sobre cómo procesar una excepción BatchUpdateException.

Para obtener información sobre avisos, utilice el método Statement.getWarnings para el objeto en el que ejecutó el método executeBatch. Luego puede obtener una descripción de error, el estado de SQL y el código de error correspondientes a cada objeto SQLWarning.

Cuando un 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:

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. SQLJ también ejecuta la sentencia que no es compatible con las sentencias del lote.
     – Cuando incluye en su programa una sentencia que no es ejecutable por lotes. En este caso, SQLJ ejecuta las sentencias que ya existen en el lote. SQLJ también ejecuta la sentencia que no es procesable por lotes.
     – Cuando después de invocar el método `ExecutionContext.setBatchLimit(n)`, añade una sentencia al lote que hace que el número de sentencias del lote sea igual o mayor que *n*. *n* puede tener uno de los valores siguientes:

       `ExecutionContext.UNLIMITED_BATCH`
       Esta constante indica que la ejecución implícita solo se produce cuando SQLJ encuentra una sentencia que es procesable por lotes pero incompatible, o que no es procesable por lotes. Establecer este valor es lo mismo que no invocar setBatchLimit.

       `ExecutionContext.AUTO_BATCH`
       Esta constante indica que la ejecución implícita se produce cuando el número de sentencias del lote alcanza un valor definido por SQLJ.

       Entero positivo
       Cuando el número de sentencias añadidas al lote alcanza este valor, SQLJ ejecuta el lote implícitamente. Sin embargo, el proceso por lotes debería ejecutarse antes de que estas muchas sentencias se hayan añadido en el caso de que SQLJ encuentre una sentencia que se pueda procesar por lotes pero que sea incompatible o bien que la sentencia no se pueda procesar por lotes.

     Para determinar el número de filas que fueron actualizadas por las sentencias de un lote que se ejecutó implícitamente, invoque el método ExecutionContext.getBatchUpdateCounts. getBatchUpdateCounts devuelve una matriz entera que contiene el número de filas que fueron actualizadas por cada sentencia del lote. El orden de los elementos de la matriz corresponde al orden en el que se añadieron las sentencias al lote. Cada elemento de la matriz puede ser uno de los valores siguientes:

     -2 Este valor indica que la sentencia de SQL se ejecutó satisfactoriamente, pero no se pudo determinar el número de filas que fueron actualizadas.

     -3 Este valor indica que la sentencia de SQL falló.

     Otro valor entero
     Este valor es el número de filas que fueron actualizadas por la sentencia.

6. Opcionalmente, una vez añadidas todas las sentencias al lote, puede inhabilitar el proceso por lotes.

   Para ello invoque el método ExecutionContext.setBatching(false). Cuando inhabilita el proceso por lotes, puede todavía ejecutar el lote implícita o explícitamente, pero no se añaden más sentencias al lote. Inhabilitar el proceso por lotes es útil cuando ya existe un lote y desea ejecutar una sentencia compatible de proceso por lotes, en lugar de añadirla al lote.

   Si desea eliminar un lote sin ejecutarlo, invoque el método ExecutionContext.cancel.
7. Si la ejecución por lotes era implícita, realice una ejecución final, explícita de executeBatch para asegurarse de que se hayan ejecutado todas las sentencias.

En el siguiente fragmento de código de programa, se asignan aumentos de salario a todos los directores mediante la ejecución de sentencias UPDATE de proceso por lotes. Los números que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
#sql iterator GetMgr(String);  // Declarar iterador de posición
{
  GetMgr deptiter;            // Declarar objeto de la clase GetMgr
  String mgrnum = null;       // Declarar variable de lenguaje principal
                              // para número de director de departamento
  int raise = 400;                 // Declarar 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
  ec.close();                    // Cerrar el contexto de ejecución
  c1.close();                    // Cerrar la conexión
}
```

Figura 42. Ejemplo de ejecución de una actualización de proceso por lotes

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, las 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.

Estos son los pasos para utilizar un iterador de nombre:

1. Declare el iterador.

   Utilice una *cláusula de declaración de iterador* para declarar un iterador de conjunto de resultados. Esto hace que se cree una clase de iterador que tiene el mismo nombre que el iterador. Para un iterador de nombre, la cláusula de declaración de iterador especifica la información siguiente:

   - El nombre del iterador
   - Una lista de nombres de columnas y tipos de datos Java
   - Información para una declaración de clase Java, tal como la indicación de si el iterador es `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 conjunto de resultados de JDBC en un iterador si el iterador está declarado como clase interna. Esta restricción no es aplicable si el iterador está declarado como clase anidada estática. Consulte ″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.

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() + " fue contratado el "
      + nameiter.HireDate());   // Usar métodos de acceso LastName y
                                // HireDate para recuperar valores de columnas
  }
  nameiter.close();             // Cerrar el iterador                    5
}
```

Figura 43. 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.

Estos son los pasos para utilizar un iterador de posición:

1. Declare el iterador.

   Utilice una *cláusula de declaración de iterador* para declarar un iterador de conjunto de resultados. Esto hace que se cree una clase de iterador que tiene el mismo nombre y atributos que el iterador. Para un iterador de posición, la cláusula de declaración de iterador especifica la información siguiente:

- El nombre del iterador
- Una lista de tipos de datos Java
- Información para una declaración de clase Java, tal como la indicación de si el iterador es `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 para otros módulos de código o paquetes. Si embargo, si especifica el iterador desde fuera de la clase anidadora debe calificar por completo el nombre de iterador con el nombre de la clase anidadora.
- Como clase interna dentro de otra clase

  Cuando declara un iterador de esta manera, puede crear una instancia del iterador solo dentro de una instancia de la clase anidadora. Sin embargo, puede declarar como públicos el iterador y otras clases contenidas en el archivo.

  No puede convertir un conjunto de resultados de JDBC en un iterador si el iterador está declarado como clase interna. Esta restricción no es aplicable si el iterador está declarado como clase anidada estática. Consulte ″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.

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 44. Ejemplo de utilización de un iterador de posición

Varios iteradores abiertos para una misma sentencia de SQL en una aplicación SQLJ

Con el 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 la pila de Java.

Los ejemplos siguientes muestran la ejecución de unas mismas operaciones en una tabla con y sin operadores abiertos simultáneamente para una misma sentencia de SQL. Estos ejemplos utilizan la siguiente declaración de iterador:

```
import java.math.*;
#sql public iterator  MultiIter(String EmpNo, BigDecimal Salary);
```

Si no se utiliza la capacidad de tener abiertos a la vez varios iteradores para una misma sentencia de SQL, si desea seleccionar valores de empleado y salario para un número de empleado determinado, debe definir una sentencia de SQL diferente para cada número de empleado, tal como muestra la Figura 45.

```
MultiIter iter1 = null;       // Instancia de iterador para
                              // recuperar datos para el primer empleado
String EmpNo1 = "000100";     // Número de empleado del primer empleado
#sql [ctx] iter2 =
  {SELECT EMPNO, SALARY FROM EMPLOYEE WHERE EMPNO = :EmpNo1};
                                    // Asignar tabla de resultados a primer iterador
MultiIter iter2 = null;       // Instancia de iterador para recuperar
                              // datos para el segundo empleado
String EmpNo2 = "000200";     // Número de empleado del segundo empleado
#sql [ctx] iter2 =
  {SELECT EMPNO, SALARY FROM EMPLOYEE WHERE EMPNO = :EmpNo2};
                                    // Asignar tabla de resultados a segundo iterador
// Procesar con iter1
// Procesar con iter2
iter1.close();                  // Cerrar los iteradores
iter2.close();
```

Figura 45. Ejemplo de operaciones de tabla simultáneas utilizando iteradores con sentencias de SQL diferentes

La Figura 46 en la página 138 muestra cómo ejecutar las mismas operaciones cuando existe la capacidad de tener abiertos a la vez varios iteradores para una misma sentencia de SQL.

```
...
MultiIter iter1 = openIter("000100"); // Invocar openIter para asignar
                                      // la tabla de resultados
                                      // (para el empleado 100) al primer
                                      // iterador
MultiIter iter2 = openIter("000200"); // Invocar openIter para asignar la
                                      // tabla de resultados al segundo
                                      // iterador.
                                      // iter1 permanece abierto cuando
                                      // se abre iter2
// Procesar con iter1
// Procesar con iter2
...
iter1.close();                  // Cerrar los iteradores
iter2.close();
...
public MultiIter openIter(String EmpNo)
                                      // Método para asignar una tabla de
                                // resultados a una instancia de iterador
{
  MultiIter iter;
  #sql [ctxt] iter =
    {SELECT EMPNO, SALARY FROM EMPLOYEE WHERE EMPNO = :EmpNo};
  return iter;                  // El método devuelve una instancia de iterador
}
```

Figura 46. Ejemplo de operaciones de tabla simultáneas utilizando iteradores con la misma sentencia de SQL

Uso de varias instancias abiertas de un iterador en una aplicación SQLJ

Pueden estar abiertas simultáneamente varias instancias de un iterador en una misma aplicación SQLJ. Una utilización de esta capacidad es abrir varias instancias de un iterador que hace uso de expresiones de lenguaje principal. Cada instancia puede utilizar un conjunto diferente de valores para una expresión de lenguaje principal.

El ejemplo siguiente muestra una aplicación con dos instancias abiertas simultáneamente de un iterador.

```
...
ResultSet myFunc(String empid)  // Método para abrir un iterador y
                                // obtener un conjunto de resultados
{
  MyIter iter;
  #sql iter = {SELECT * FROM EMPLOYEE WHERE EMPNO = :empid};
  return iter.getResultSet();
}

// Una aplicación puede invocar este método para obtener un conjunto
// de resultados para cada ID de empleado. La aplicación puede
// procesar cada conjunto de resultados por separado.
...
ResultSet rs1 = myFunc("000100");  // Obtener registro del ID de empleado 000100
...
ResultSet rs2 = myFunc("000200");  // Obtener registro del ID de empleado 000200
```

Figura 47. Ejemplo de apertura de más de una instancia de un iterador en una misma aplicación

Al igual que ocurre con cualquier otro iterador, es necesario cerrar este iterador cuando termine de utilizarlo para evitar un uso excesivo de espacio de almacenamiento.

Utilización de iteradores desplazables en una aplicación SQLJ

Además de avanzar fila a fila por una tabla de resultados, puede desear ir hacia atrás o directamente a una fila determinada. El IBM Data Server Driver para JDBC y SQLJ proporciona esta capacidad.

Un iterador en el que se puede desplazar hacia delante, hacia atrás o hasta una fila determinada se denomina *iterador desplazable*. Un iterador desplazable de SQLJ 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 excepción de SQL.

Si un iterador desplazable es dinámico, el tamaño de la tabla de resultados y el orden de las filas en la tabla de resultados pueden cambiar después de abrir el iterador. Las filas que se insertan o suprimen con sentencias INSERT y DELETE ejecutadas por el mismo proceso de aplicación son visibles inmediatamente. Las filas que se insertan o suprimen con sentencias INSERT y DELETE ejecutadas por otros procesos de aplicación son visibles una vez confirmados los cambios.

Importante: Los servidores DB2 Database para Linux, UNIX y Windows no son compatibles con 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.

Para crear y utilizar un iterador desplazable, debe seguir estos pasos:

1. Especifique una cláusula de declaración de iterador que incluya las cláusulas siguientes:
   - `implements sqlj.runtime.Scrollable`

     Esto indica que el iterador es desplazable.
   - `with (sensitivity=INSENSITIVE|SENSITIVE|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 de nombre o de posición. Por ejemplo, la siguiente cláusula de declaración de iterador declara un iterador de posición que es sensible, dinámico y desplazable:

```
#sql public iterator ByPos
  implements sqlj.runtime.Scrollable
  with (sensitivity=SENSITIVE, dynamic=true)  (String);
```

La siguiente cláusula de declaración de iterador declara un iterador de nombre que es insensible y desplazable:

```
#sql public iterator ByName
  implements sqlj.runtime.Scrollable
  with (sensitivity=INSENSITIVE) (String EmpNo);
```

Restricción: No puede utilizar un iterador desplazable para seleccionar columnas con los tipos de datos siguientes en una tabla de un servidor DB2 Database para Linux, UNIX y Windows:

- LONG VARCHAR
- LONG VARGRAPHIC
- BLOB
- CLOB
- Un tipo diferenciado que esté basado en cualquiera de los tipos de datos anteriores de esta lista
- Un tipo estructurado

2. Cree un objeto de iterador, que es una instancia de la clase de iterador utilizada.
3. Si desea indicar al entorno de ejecución de SQLJ la dirección inicial de la recuperación de datos, utilice el método `setFetchDirection(int dirección)`. *dirección* puede ser `FETCH_FORWARD` o `FETCH_REVERSE`. Si no invoca setFetchDirection, la dirección de recuperación de datos es `FETCH_FORWARD`.
4. Para cada fila a la que desee acceder:

   En el caso de un iterador con nombre, siga estos pasos:

   a. Sitúe el cursor utilizando uno de los métodos listados en la tabla siguiente.

Tabla 19. Métodos sqlj.runtime.Scrollable para situar un cursor desplazable

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

Tabla 19. Métodos sqlj.runtime.Scrollable para situar un cursor desplazable (continuación)

Método	Sitúa el cursor

Notas:

1. Si el cursor está situado después de la última fila de la tabla de resultados, este método posiciona el cursor en la última fila.
2. Si el valor absoluto de *n* es mayor que el número de filas de la tabla de resultados, este método posiciona el cursor después de la última fila si *n* es positivo, o antes de la primera fila si *n* es negativo.
3. Suponga que *m* es el número de filas de la tabla de resultados y *x* es el número de fila actual de la tabla de resultados. Si *n*>0 y *x*+*n*>*m*, el iterador se sitúa 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 20 lista las cláusulas que puede utilizar para posicionar el cursor.

Tabla 20. Cláusulas de FETCH para situar un cursor desplazable

Método	Sitúa el cursor
FIRST	En la primera fila de la tabla de resultados
LAST	En la última fila de la tabla de resultados
PRIOR[1]	En la fila anterior de la tabla de resultados
NEXT	En la fila siguiente de la tabla de resultados
ABSOLUTE(*n*)[2]	Si *n*>0, en la fila *n* de la tabla de resultados. Si *n*<0 y *m* es el número de filas de la tabla de resultados, sitúa el iterador en la fila *m*+*n*+*1* de la tabla de resultados.
RELATIVE(*n*)[3]	Si *n*>0, en la fila que está *n* filas después de la fila actual. Si *n*<0, en la fila que está situada *n* filas antes de la fila actual. Si *n*=0, en la fila actual.
AFTER[4]	Después de la última fila de la tabla de resultados
BEFORE[4]	Antes de la primera fila de la tabla de resultados

Notas:

1. Si el cursor está situado después de la última fila de la tabla de resultados, este método posiciona el cursor en la última fila.
2. Si el valor absoluto de *n* es mayor que el número de filas de la tabla de resultados, este método posiciona el cursor después de la última fila si *n* es positivo, o antes de la primera fila si *n* es negativo.
3. Suponga que *m* es el número de filas de la tabla de resultados y *x* es el número de fila actual de la tabla de resultados. Si *n*>0 y *x*+*n*>*m*, el iterador se sitúa a continuación de la última fila. Si *n*<0 y *x*+*n*<1, el iterador se sitúa antes de la primera fila.
4. No se asignan valores a las expresiones de lenguaje principal.

b. Si las operaciones de actualización o supresión realizadas por el iterador o por otros medios son visibles en la tabla de resultados, invoque el método getWarnings para determinar si la fila actual es un hueco.

5. Invoque el método close para cerrar el iterador.

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  iterator ScrollIter implements sqlj.runtime.Scrollable          1
  (String EmpNo, String LastName);
{
  ScrollIter scrliter;                                                2
  #sql [ctxt]
  scrliter={SELECT EMPNO, LASTNAME FROM EMPLOYEE};
  scrliter.afterLast();
  while (scrliter.previous())                                         4a
  {
    System.out.println(scrliter.EmpNo() + " "                         4c
      + scrliter.LastName());
  }
  scrliter.close();                                                   5
}
```

Figura 48. Ejemplo de utilización de iteradores desplazables

Invocación de procedimientos almacenados en una aplicación SQLJ

Para invocar un procedimiento almacenado, utilice una cláusula ejecutable que contenga una sentencia CALL de SQL.

Puede ejecutar la sentencia CALL con parámetros de identificador de lenguaje principal. 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:

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.

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 49. Ejemplo de invocación de un procedimiento almacenado en una aplicación SQLJ

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

Algunos procedimientos almacenados devuelven uno o más conjuntos de resultados al programa solicitante. 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 contexto de ejecución para recuperar el conjunto de resultados.
   b. Si no conoce el contenido del conjunto de resultados, utilice el método ResultSetMetaData para obtener esta información.
   c. Utilice un iterador de conjunto de resultados de SQLJ o un conjunto de resultados de JDBC para recuperar las filas del conjunto de resultados.

Los conjuntos de resultados se devuelven al programa solicitante en el mismo orden en el que se abren los cursores del programa en el procedimiento almacenado. Cuando no existen más conjuntos de resultados para recuperar, getNextResultSet devuelve un valor nulo.

Existen dos formatos de getNextResultSet:

```
getNextResultSet();
getNextResultSet(int actual);
```

Cuando invoca el primer formato de getNextResultSet, SQLJ cierra el conjunto de resultados que está abierto actualmente y pasa al conjunto de resultados siguiente.

Cuando invoca el segundo formato de getNextResultSet, el valor de *actual* indica lo que SQLJ realiza con el conjunto de resultados abierto actualmente antes de pasar al conjunto de resultados siguiente:

java.sql.Statement.CLOSE_CURRENT_RESULT
Especifica que el objeto ResultSet actual se cierra cuando se devuelve el objeto ResultSet siguiente.

java.sql.Statement.KEEP_CURRENT_RESULT
Especifica que el objeto ResultSet actual permanece abierto cuando se devuelve el objeto ResultSet siguiente.

java.sql.Statement.CLOSE_ALL_RESULTS
Especifica que todos los objetos ResultSet abiertos se cierran cuando se devuelve el objeto ResultSet siguiente.

El 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 que aparecen a la derecha de algunas sentencias corresponden a los pasos descritos anteriormente.

```
ExecutionContext execCtx=myConnCtx.getExecutionContext();      1
#sql [myConnCtx, execCtx] {CALL MULTRSSP()};                   2
           // MULTRSSP devuelve varios conjuntos de resultados
ResultSet rs;
while ((rs = execCtx.getNextResultSet()) != null)              3a
{
  ResultSetMetaData rsmeta=rs.getMetaData();                   3b
  int numcols=rsmeta.getColumnCount();
  while (rs.next())                                            3c
  {
    for (int i=1; i<=numcols; i++)
    {
      String colval=rs.getString(i);
      System.out.println("Columna " + i + "valor es " + colval);
    }
  }
}
```

Figura 50. Recuperación de conjuntos de resultados de un procedimiento almacenado

Objetos LOB en aplicaciones SQLJ con IBM Data Server Driver para JDBC y SQLJ

Con el 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 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 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. Estas tareas adicionales de proceso pueden afectar al rendimiento del sistema.

Cuando el controlador JDBC no puede determinar inmediatamente el tipo de datos de un parámetro que se utiliza con una columna LOB, es necesario elegir un tipo de datos para el parámetro que sea compatible con el tipo de datos LOB.

Parámetros de entrada para columnas BLOB

Para los parámetros de entrada de columnas BLOB, 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. Para esta llamada es necesario especificar la longitud exacta de los datos de entrada:

  ```
  java.io.ByteArrayInputStream byteStream =
    new java.io.ByteArrayInputStream(byteData);
  int numBytes = byteData.length;
  sqlj.runtime.BinaryStream binStream =
    new sqlj.runtime.BinaryStream(byteStream, numBytes);
  #sql {CALL STORPROC(:IN binStream)};
  ```

  No puede utilizar esta técnica para parámetros de entrada/salida.

Parámetros de salida para columnas BLOB

Para los parámetros de salida o entrada/salida de columnas BLOB, puede utilizar la técnica siguiente:

- Declare el parámetro de salida o variable de entrada/salida con un tipo de datos java.sql.Blob:

  ```
  java.sql.Blob blobData = null;
  #sql CALL STORPROC (:OUT blobData)};
  ```

  ```
  java.sql.Blob blobData = null;
  #sql CALL STORPROC (:INOUT blobData)};
  ```

Parámetros de entrada para columnas CLOB

Para los parámetros de entrada de columnas CLOB, puede utilizar una de las técnicas siguientes:

- Utilice una variable de entrada java.sql.Clob, que se corresponde exactamente con una columna CLOB:

```
#sql CALL STORPROC(:IN clobData)};
```

Para poder utilizar una variable de entrada java.sql.Clob, es necesario crear un objeto java.sql.Clob y luego llenar con datos ese objeto.

- Utilice uno de los tipos siguientes de parámetros de entrada:
  - Un parámetro de entrada sqlj.runtime.CharacterStream:

    ```
    java.lang.String charData;
    java.io.StringReader reader = new java.io.StringReader(charData);
    sqlj.runtime.CharacterStream charStream =
      new sqlj.runtime.CharacterStream (reader, charData.length);
    #sql {CALL STORPROC(:IN charStream)};
    ```

  - Un parámetro sqlj.runtime.UnicodeStream, para datos Unicode UTF-16:

    ```
    byte[] charDataBytes = charData.getBytes("UnicodeBigUnmarked");
    java.io.ByteArrayInputStream byteStream =
      new java.io.ByteArrayInputStream(charDataBytes);
    sqlj.runtime.UnicodeStream uniStream =
      new sqlj.runtime.UnicodeStream(byteStream, charDataBytes.length );
    #sql {CALL STORPROC(:IN uniStream)};
    ```

  - Un parámetro sqlj.runtime.AsciiStream, para datos ASCII:

    ```
    byte[] charDataBytes = charData.getBytes("US-ASCII");
    java.io.ByteArrayInputStream byteStream =
      new java.io.ByteArrayInputStream (charDataBytes);
    sqlj.runtime.AsciiStream asciiStream =
      new sqlj.runtime.AsciiStream (byteStream, charDataBytes.length);
    #sql {CALL STORPROC(:IN asciiStream)};
    ```

  Para estas llamadas es necesario especificar la longitud exacta de los datos de entrada. No puede utilizar esta técnica para parámetros de entrada/salida.
- Utilice un parámetro de entrada java.lang.String:

  ```
  java.lang.String charData;
  #sql {CALL STORPROC(:IN charData)};
  ```

Parámetros de salida para columnas CLOB

Para los parámetros de salida o entrada/salida de columnas CLOG, 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 entrada/salida 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 conexión JDBC para crear contexto de conexión SQLJ u obtenga una conexión JDBC a partir de un contexto de conexión SQLJ.
- Utilice un iterador SQLJ para obtener datos a partir de un 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 JDBC para obtener una conexión JDBC.
4. Invoque el constructor ConnectionContext con el objeto Connection como argumento para crear el objeto ConnectionContext.

Obtención de una conexión JDBC a partir de una contexto de conexión SQLJ - Siga estos pasos:

1. Ejecute una cláusula de declaración de conexión SQLJ para crear una clase ConnectionContext.
2. Cargue el controlador u obtenga una instancia de DataSource.
3. Invoque el constructor ConnectionContext con el URL del controlador y cualquier otro parámetro necesario como argumentos para crear el objeto ConnectionContext.
4. Invoque el método ConnectionContext.getConnection de JDBC para crear el objeto Connection de JDBC.

Consulte ″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 51 en la página 148 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 51. Conversión de un conjunto de resultados JDBC en un iterador SQLJ

Notas para la Figura 51:

Nota	Descripción
1	Este cláusula de SQLJ crea la clase de iterador de nombre ByName, cuyos métodos accesores LastName() y HireDate() obtienen datos de las columnas LASTNAME y HIREDATE de la tabla de resultados.
2	Esta sentencia y las dos sentencias que le siguen crean y preparan una consulta para su ejecución dinámica mediante JDBC.
3	Esta sentencia de JDBC ejecuta la sentencia SELECT y asigna la tabla de resultados al conjunto de resultados rs.
4	Esta cláusula de conversión a iterador convierte el 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 convertir un iterador en un conjunto de resultados, necesita recuperar filas utilizando solamente el conjunto de resultados.

El código mostrado en la Figura 52 en la página 149 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 52. Conversión de un iterador SQLJ en un conjunto de resultados JDBC

Notas para la Figura 52:

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() convierte el iterador iter en el ResultSet rs.
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` del 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 objeto ResultSet generado o el iterador subyacente al final del programa.

  Cuando cierra el objeto iterador utilizado para crear un objeto ResultSet, también se cierra el objeto ResultSet. Cuando cierra el objeto ResultSet generado, también se cierra el objeto iterador. En general, es mejor cerrar el objeto que se ha utilizado en último lugar.
- Para el IBM Data Server Driver para JDBC y SQLJ, que da soporte a iteradores desplazables y a conjuntos de resultados desplazables y actualizables, son aplicables las restricciones siguientes:
  - Los iteradores desplazables tienen las mismas restricciones que sus conjuntos de resultados de JDBC subyacentes.
  - 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.

Siga estos pasos para utilizar métodos de ExecutionContext:

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.

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 que aparecen a la derecha de algunas sentencias corresponden a los 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 IBM Data Server Driver para JDBC y SQLJ

DB2 para z/OS y DB2 para i5/OS permiten la utilización del tipo de datos ROWID para una columna de una tabla. Un ROWID es un valor que identifica una fila de una tabla de una forma exclusiva.

Si utiliza valores ROWID en programas SQLJ, es necesario que personalice esos programas.

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 ");
     printBytes(rowid.getBytes());
                           // Utilizar el método getBytes para convertir
                           // el valor a bytes para su impresión
      #sql {FETCH :positrowid INTO :id, :i_fmt, :rowid};
                                // Recuperar la fila siguiente
    }
  positrowid.close();           // Cerrar el iterador
}
```

Figura 53. Ejemplo de utilización de un iterador para recuperar valores 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 ROWID de entrada,
                                   // de salida y de entrada/salida
...
#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 de salida ");
printBytes(out_rowid.getBytes());
                           // Utilizar el método getBytes para convertir
                           // el valor a bytes para su impresión
System.out.println("Valor de parámetro de entrada/salida ");
printBytes(inout_rowid.getBytes());
```

Figura 54. Ejemplo de invocación de un procedimiento almacenado con un parámetro ROWID

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 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:

```
String empNumVar;
int shoeSizeVar;
...
#sql [myConnCtx] {CREATE DISTINCT TYPE SHOESIZE AS INTEGER WITH COMPARISONS};
                            // Crear tipo diferenciado
#sql [myConnCtx] {COMMIT}; // Confirmar la creación
#sql [myConnCtx] {CREATE TABLE EMP_SHOE
  (EMPNO CHAR(6), EMP_SHOE_SIZE SHOESIZE)};
                            // Crear tabla utilizando tipo diferenciado
#sql [myConnCtx] {COMMIT}; // Confirmar la creación
#sql [myConnCtx] {INSERT INTO EMP_SHOE
  VALUES('000010',6)};      // Insertar una fila en la tabla
#sql [myConnCtx] {COMMIT}; // Confirmar la inserción
#sql [myConnCtx] {SELECT EMPNO, EMP_SHOE_SIZE
  INTO :empNumVar, :shoeSizeVar
  FROM EMP_SHOE};           // Recuperar la fila
System.out.println("Employee number: " + empNumVar +
  " Shoe size: " + shoeSizeVar);
```

Figura 55. Definición y utilización de un tipo diferenciado

Invocación de procedimientos almacenados con parámetros ARRAY en aplicaciones SQLJ

Las aplicaciones SQLJ que se ejecutan bajo el IBM Data Server Driver para JDBC y SQLJ pueden invocar procedimientos almacenados que tienen parámetros ARRAY.

Puede utilizar objetos java.sql.Array como parámetros IN, OUT o INOUT en un procedimiento almacenado.

Utilice el método DB2Connection.createArrayOf para asignar valores a un parámetro de entrada ARRAY de un procedimiento almacenado.

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;
                                    // Defina 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 = rs.getString(2); // Obtener número de teléfono
  System.out.println("Número de teléfono = " + phoneNum);
}
```

Puntos de salvaguarda en aplicaciones SQLJ

Cuando se utiliza el 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 56. 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 interno XML para almacenar datos XML en una columna en forma de conjunto estructurado de nodos en formato de árbol.

En las aplicaciones, los datos XML se encuentran en formato de serie serializada.

En las aplicaciones SQLJ, se puede realizar lo siguiente:

- Almacenar un documento XML completo en una columna XML mediante sentencias INSERT o UPDATE.
- 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.

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 de SQLJ

Cuando actualiza o inserta datos en columnas XML de una tabla en una aplicación SQLJ, los datos de entrada deben tener el formato de serie de caracteres serializada.

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 IBM Data Server Driver para JDBC y SQLJ Versión 4.0 o posterior)
- com.ibm.db2.jcc.DB2Xml (en desuso)
- String
- byte
- Blob
- Clob
- sqlj.runtime.AsciiStream
- sqlj.runtime.BinaryStream
- sqlj.runtime.CharacterStream

Para los tipos de corrientes, debe utilizarse la expresión de lenguaje principal sqlj.runtime.*tipo*Stream, en lugar de la expresión de lenguaje principal java.io.*tipo*InputStream, de modo que se pueda pasar la longitud de la corriente al controlador JDBC.

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 IBM Data Server Driver para JDBC y SQLJ Versión 4.0 o posterior)
- com.ibm.db2.jcc.DB2Xml (en desuso)
- String
- 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 ocom.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 correspondientes tipos de datos de salida y el tipo de codificación utilizado en las declaraciones XML.

Tabla 21. 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	Ninguno
SQLXML.getString	String	Ninguno
DB2Xml.getDB2AsciiStream	InputStream	Ninguno
DB2Xml.getDB2BinaryStream	InputStream	Ninguno
DB2Xml.getDB2Bytes	byte[]	Ninguno
DB2Xml.getDB2CharacterStream	Reader	Ninguno
DB2Xml.getDB2String	String	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	String	ISO-10646-UCS-2

Si la aplicación ejecuta la función XMLSERIALIZE para los datos que se deben devolver, después de ejecutar la función, el tipo de los datos es el especificado en la función XMLSERIALIZE, no el tipo de datos XML. Por tanto, el controlador maneja los datos de acuerdo con el tipo especificado y no tiene en cuenta ninguna 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.

Utilización en SQLJ de funciones del SDK de Java Versión 5

Las aplicaciones SQLJ puede utilizar varias funciones que aparecieron con 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; también pueden 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 aporta el tipo `enum` para los tipos enumerados.

No puede incluir los tipos `enum` de Java directamente en los programas SQLJ, pero puede incluirlos 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 = null;
  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 puede 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.

La tabla siguiente muestra los valores que puede especificar en la cláusula SET TRANSACTION ISOLATION LEVEL y sus valores equivalentes en DB2.

Tabla 22. 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

Tabla 22. Niveles de aislamiento equivalentes en SQLJ y DB2 (continuación)

Valor de SET TRANSACTION	Nivel de aislamiento en DB2
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.

Para ello utilizará cláusulas de ejecución que contienen sentencias COMMIT o ROLLBACK de SQL.

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.

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.

- 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.

Para manejar errores de SQL en aplicaciones SQLJ, siga estos pasos:

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.

Para el controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows (controlador JDBC de DB2 de tipo 2), utilice el SQLException para recuperar información sobre el error de SQL.

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 de DB2 no emiten excepciones de SQL. Para manejar los avisos de DB2, es necesario otorgar al programa acceso a la clase java.sql.SQLWarning.

Si desea recuperar información específica de DB2 sobre un aviso, es necesario también que otorgue al programa acceso a la interfaz com.ibm.db2.jcc.DB2Diagnosable y a la clase com.ibm.db2.jcc.DB2Sqlca. Luego siga estos pasos:

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 de DB2, invoque el método getWarnings después de ejecutar una cláusula de SQLJ.

   getWarnings devuelve el primer objeto SQLWarning generado por una sentencia de SQL. Los objetos SQLWarning subsiguientes se encadenan al primero de ellos.
3. Para recuperar información específica de DB2 contenida en el objeto SQLWarning utilizando el IBM Data Server Driver para JDBC y SQLJ, siga las instrucciones de la sección ″Manejo de una excepción de SQL cuando se utiliza el IBM Data Server Driver para JDBC y SQLJ″.

El ejemplo siguiente muestra cómo recuperar un objeto SQLWarning para una cláusula de SQL cuyo contexto de ejecución es execCtx. Los números situados a la derecha de determinadas sentencias corresponden a 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 manera se liberan inmediatamente los recursos del objeto de contexto de conexión de DB2 y SQLJ.

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.

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 el IBM Data Server Driver para JDBC y SQLJ, puede seleccionar un mecanismo de seguridad especificando un valor para la propiedad `securityMechanism`.

Puede definir esta propiedad de una de las maneras siguientes:

- Si utiliza la interfaz DriverManager, defina `securityMechanism` en un objeto java.util.Properties antes de invocar la modalidad del método getConnection que hace uso del parámetro java.util.Properties.
- Si utiliza la interfaz DataSource, y está creando y desplegando sus propios objetos DataSource, invoque el método DataSource.setSecurityMechanism después de crear un objeto DataSource.

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 23. Soporte de servidor de bases de datos para mecanismos de seguridad de IBM Data Server Driver para JDBC y SQLJ

Mecanismo de seguridad	**Válido para**		
	DB2 Database para Linux, UNIX y Windows	DB2 para z/OS	IBM Informix Dynamic Server
ID de usuario y contraseña	Sí	Sí	Sí
ID de usuario solamente	Sí	Sí	Sí
ID de usuario y contraseña cifrada	Sí	Sí	Sí
ID de usuario cifrado	Sí	Sí	No
ID de usuario cifrado y contraseña cifrada	Sí	Sí	Sí
ID de usuario cifrado y datos cifrados sensibles a la seguridad	No	Sí	No
ID de usuario cifrado, contraseña cifrada y datos cifrados sensibles a la seguridad	Sí	Sí	No
Kerberos[1]	Sí	Sí	No
Plugin[1]	Sí	No	No

Nota:

1. Disponible solo para el IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 4.

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, el 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 24. Mecanismos de seguridad soportados por el 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	`DB2BaseDataSource.ENCRYPTED_PASSWORD_SECURITY`
ID de usuario cifrado	`DB2BaseDataSource.ENCRYPTED_USER_ONLY_SECURITY`
ID de usuario cifrado y contraseña cifrada	`DB2BaseDataSource.ENCRYPTED_USER_AND_PASSWORD_SECURITY`
ID de usuario cifrado y datos cifrados sensibles a la seguridad	`DB2BaseDataSource.ENCRYPTED_USER_AND_DATA_SECURITY`
ID de usuario cifrado, contraseña cifrada y datos cifrados sensibles a la seguridad	`DB2BaseDataSource.ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY`
Kerberos	`DB2BaseDataSource.KERBEROS_SECURITY`
Plugin	`DB2BaseDataSource.PLUGIN_SECURITY`

La tabla siguiente muestra los tipos posibles de autenticación 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 25. Tipos de autenticación del servidor DB2 Database para Linux, UNIX y Windows y valores de securityMechanism del IBM Data Server Driver para JDBC y SQLJ compatibles

Tipo de autenticación 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

Tabla 25. Tipos de autenticación del servidor DB2 Database para Linux, UNIX y Windows y valores de securityMechanism del IBM Data Server Driver para JDBC y SQLJ compatibles (continuación)

Tipo de autenticación del servidor DB2 Database para Linux, UNIX y Windows	Valor de securityMechanism
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 ID de usuario y 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";
                           // Definir URL para fuente de datos

Connection con = DriverManager.getConnection(url, id, pw);
                           // Crear conexión
```

Otro método consiste en definir el ID de usuario y la contraseña directamente en la serie de caracteres del URL. Por ejemplo:

```
import java.sql.*;         // Base JDBC
...
String url =
  "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose:user=dbadm;password=dbadm;";

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

Como alternativa, puede establecer el ID de usuario y la contraseña definiendo las propiedades `user` y `password` en un objeto Properties, y luego invocar la modalidad del método getConnection que hace uso del objeto Properties como parámetro. Opcionalmente, puede definir la propiedad `securityMechanism` para indicar que está utilizando la seguridad basada en un ID de usuario y contraseña. Por ejemplo:

```
import java.sql.*;         // Base JDBC
import com.ibm.db2.jcc.*;                 // Implementación de 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
                                    // basado en ID de usuario y contraseña
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                                    // Definir URL para fuente de datos
Connection con = DriverManager.getConnection(url, properties);
                                    // Crear conexión
```

Para la interfaz DataSource: puede especificar el ID de usuario y la contraseña directamente en la invocación de DataSource.getConnection. Por ejemplo:

```
import java.sql.*;          // Base JDBC
import com.ibm.db2.jcc.*;              // Implementación de 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
                                           // basado en ID de usuario y contraseña
```

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

Para la interfaz DataSource: si crea y despliega el objeto DataSource, establezca el ID de usuario y el mecanismo de seguridad invocando los métodos DataSource.setUser y DataSource.setSecurityMechanism después de crear el objeto DataSource. Por ejemplo:

```
import java.sql.*;                      // 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");
                                        // Definir nombre del 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
```

Seguridad por contraseña cifrada, seguridad por ID de usuario cifrado o seguridad por ID de usuario cifrado y contraseña cifrada cuando se utiliza IBM Data Server Driver para JDBC y SQLJ

IBM Data Server Driver para JDBC y SQLJ es compatible con la seguridad por contraseña cifrada, la seguridad por ID de usuario cifrado o la contraseña por ID de usuario cifrado y contraseña cifrada para acceder a fuentes de datos.

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.

Es necesario IBM JCE para utilizar comunicaciones cifradas cliente/servidor de 256 bits entre el controlador IBM Data Server Driver para JDBC y SQLJ y servidores DB2 Database para Linux, UNIX y Windows.

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 solamente para conexiones con servidores DB2 para z/OS.

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 mecanismo de seguridad basado en un ID de usuario y una contraseña cifrada:

```
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_PASSWORD_SECURITY +
  ""));
                                         // Establecer mecanismo de seguridad como
                                         // de ID usuario y contraseña cifrada
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                                         // Establecer URL para fuente de datos
Connection con = DriverManager.getConnection(url, properties);
                                         // Crear la conexión
```

Para la interfaz DataSource: si crea y despliega el objeto DataSource, puede definir el 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 mecanismo de seguridad basado en un ID de usuario cifrado y una contraseña cifrada:

```
import java.sql.*;                          // Base JDBC
import com.ibm.db2.jcc.*;                // Implementación de 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_PASSWORD_SECURITY);
                                        // Establecer mecanismo de seguridad como
                                        // de ID usuario y contraseña cifrada
```

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 el IBM Data Server Driver para conectividad JDBC y SQLJ 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 autenticación 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 base de datos.

Es necesario que defina las propiedades `user`, `password`, `kerberosServerPrincipal` y `securityMechanism`. La propiedad `kerberosServerPrincipal` especifica el nombre de principal que el servidor de base 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.*;                          // 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("password", "db2adm");     // Definir 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";
```

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

Para la interfaz DataSource: si crea y despliega el objeto DataSource, establezca el servidor Kerberos y el mecanismo de seguridad invocando los métodos DataSource.setKerberosServerPrincipal y `DataSource.setSecurityMechanism` después de crear el objeto DataSource. Por ejemplo:

```
import java.sql.*;                      // 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 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");
                                        // Definir nombre del 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 base de datos.

Es necesario establecer las propiedades `kerberosServerPrincipal` y `securityMechanism`.

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

Para la interfaz DataSource: si crea y despliega el objeto DataSource, establezca el servidor Kerberos y el mecanismo de seguridad invocando los métodos DataSource.setKerberosServerPrincipal y `DataSource.setSecurityMechanism` después de crear el objeto DataSource. Por ejemplo:

```
import java.sql.*;                          // JDBC base
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");
                                            // Definir nombre del 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 base de datos mediante una credencial delegada que le pasa otro principal.

Es necesario establecer las propiedades `kerberosServerPrincipal`, `gssCredential` , y `securityMechanism`.

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

Para la interfaz DataSource: Si crea y despliega el objeto DataSource, 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, el 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 conectividad JDBC y SQLJ 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 el 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 el 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 26. 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 conexión.
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 un 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);
```

Soporte de contexto fiable de IBM Data Server Driver para JDBC y SQLJ

El IBM Data Server Driver para JDBC y SQLJ proporciona métodos que permiten establecer y utilizar conexiones fiables en programas Java.

Se pueden utilizar conexiones fiables en IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 4 con DB2 Database para Linux, UNIX y Windows y DB2 para z/OS, y IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 2 con DB2 para z/OS.

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 DB2, el servidor de middleware puede utilizar esa conexión de base de datos bajo un usuario diferente sin volver a autenticar 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, el 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 un cookie exclusivo para la instancia de conexión. El cookie es generado por el controlador JDBC y se utiliza para la autenticación 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:

- El cookie de la conexión inicial
- Nuevas propiedades de conexión para la conexión reutilizada

El controlador JDBC comprueba que el cookie proporcionado coincida con el 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 el 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 el 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");
// Segundo elemento obtenido de llamada anterior a getDB2TrustedPooledConnection
// es el cookie de la conexión. Convertirlo en una 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 dato 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 IBM Data Server Driver para JDBC y SQLJ proporciona soporte para Security Socket Layer (SSL) mediante Java Secure Socket Extension (JSSE).

Puede utilizar soporte de SSL en sus aplicaciones Java si utiliza IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 4 con DB2 para z/OS Versión 9.1 o posterior, o con DB2 Database para Linux, UNIX y Windows Versión 9.1, Fix Pack 2 o posterior.

En el IBM Data Server Driver para JDBC y SQLJ, habilite el soporte de SSL estableciendo la propiedad DB2BaseDataSource.sslConnection en `true`. Esto hace que el IBM Data Server Driver para JDBC y SQLJ utilice un socket SSL para conectar con servidores de bases de datos DB2.

Para que una aplicación cliente pueda utilizar SSL, el usuario de la aplicación cliente debe:

- Obtener un certificado del administrador de seguridad del sistema principal y utilizar el programa de utilidad Java keytool para importar el certificado a un almacén de confianza.

  Por ejemplo, suponga que se ha recuperado un certificado a partir de un servidor de bases de datos DB2 y se ha almacenado en un archivo llamado 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
  ```

- Establezca las siguientes propiedades del sistema Java:

  javax.net.ssl.trustStore
  Especifica el almacén de confianza.

  javax.net.ssl.trustStorePassword
  Especifica la contraseña del almacén de confianza.

El IBM Data Server Driver para JDBC y SQLJ puede utilizar el proveedor compatible con FIPS de IBM para SSL o el proveedor SunJSSE. Para utilizar el IBM Data Server Driver para JDBC y SQLJ en modalidad compatible con FIPS, utilice los proveedores aprobados por FIPS IBMJSSEFIPSProvider e IBMJCEFIPS o habilite la modalidad FIPS en el proveedor IBMJSSE2.

Para utilizar los proveedores IBMJSSEFIPSProvider e IBMJCEFIPS añada las líneas siguientes al archivo `java.security`:

```
security.provider.n=com.ibm.fips.jsse.IBMJSSEFIPSProvider
security.provider.n=com.ibm.crypto.fips.provider.IBMJCEFIPS

ssl.SocketFactory.provider=com.ibm.fips.jsse.JSSESocketFactory
ssl.ServerSocketFactory.provider= com.ibm.fips.jsse.JSSEServerSocketFactory
```

n es el orden de preferencia. Los proveedores especificados anteriormente deben tener un orden de preferencia superior (valor inferior *n*) a los proveedores que no sean FIPS en el archivo `java.security`.

Para habilitar la modalidad FIPS en el proveedor IBMJSSE2, siga los pasos que se indican a continuación:

1. Establezca la propiedad del sistema FIPS `IBMJSSE2` para habilitar la propiedad FIPS:

```
com.ibm.jsse2.JSSEFIPS=true
```

2. Establezca las propiedades de seguridad para garantizar que todo el código de JSSE utilice el proveedor IBMJSSE2:

   ```
   ssl.SocketFactory.provider=com.ibm.jsse2.SSLSocketFactoryImpl
   ssl.SocketFactory.provider=com.ibm.jsse2.SSLServerSocketFactoryImpl
   ```

3. Añada el proveedor criptográfico IBMJCEFIPS.

   Para hacerlo, añada com.ibm.crypto.fips.provider.IBMJCEFIPS a la lista de proveedores antes del proveedor IBMJCE. No elimine el proveedor IBMJCE. El proveedor IBMJCE es necesario para el soporte del almacén de claves.

No es necesario realizar cambios en las aplicaciones para que el proveedor IBMJSSE2 se ejecute en modalidad FIPS.

Restricciones:

- Las aplicaciones JSSE que no estén de modalidad FIPS no se pueden ejecutar en una JVM que esté en modalidad FIPS.
- Cuando el proveedor IBMJSSE2 se ejecuta en modalidad FIPS, no puede utilizar la criptografía por hardware.

Seguridad para preparar aplicaciones SQLJ con IBM Data Server Driver para JDBC y SQLJ

Dos formas de proporcionar seguridad durante la preparación de aplicaciones SQLJ es permitir a los usuarios solamente personalizar las aplicaciones y limitar el acceso a un determinado conjunto de tablas durante la personalización.

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 (sistema de vincular y ejecutar). 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.

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. Seguridad cuando se utiliza el controlador JDBC de DB2 de tipo 2

El controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows (controlador JDBC de DB2 de tipo 2) es compatible con la seguridad por ID de usuario y contraseña.

Debe establecer el ID de usuario y la contraseña o no definir ninguno de ellos. Si no define un ID de usuario y contraseña, el controlador utiliza el ID de usuario y contraseña del usuario que está conectado actualmente al sistema operativo.

Para especificar la seguridad basada en un ID de usuario y una contraseña para una conexión JDBC, utilice una de las técnicas siguientes.

Para la interfaz DriverManager: puede especificar el ID de usuario y la contraseña directamente en la invocación de DriverManager.getConnection. Por ejemplo:

```
import java.sql.*;         // Base JDBC
...
String id = "db2adm";                  // Definir ID de usuario
Sring pw = "db2adm";                   // Definir contraseña
String url = "jdbc:db2:toronto";
                           // Establecer URL para fuente de datos
Connection con = DriverManager.getConnection(url, id, pw);
                           // Crear conexión
```

Como alternativa, puede establecer el ID de usuario y la contraseña definiendo las propiedades `user` y `password` en un objeto Properties, y luego invocar la modalidad del método getConnection que hace uso del objeto Properties como parámetro. Por ejemplo:

```
import java.sql.*;         // Base JDBC
import COM.ibm.db2.jdbc.*;                // Implementación de JDBC para
                                          // controlador JDBC de DB2 de tipo 2
...
Properties properties = new java.util.Properties();
                                          // Crear objeto Properties
properties.put("user", "db2adm");         // Establecer ID usuario para conexión
properties.put("password", "db2adm");     // Establecer contraseña para conexión
String url = "jdbc:db2:toronto";
                                          // Establecer URL para fuente de datos
Connection con = DriverManager.getConnection(url, properties);
                                          // Crear conexión
```

Para la interfaz DataSource: puede especificar el ID de usuario y la contraseña directamente en la invocación de DataSource.getConnection. Por ejemplo:

```
import java.sql.*;         // Base JDBC
import COM.ibm.db2.jdbc.*;                // Implementación de JDBC para
                                          // controlador JDBC de DB2 de tipo 2
...
Context ctx=new InitialContext();      // Crear contexto para JNDI
DataSource ds=(DataSource)ctx.lookup("jdbc/sampledb");
                                          // Obtener objeto DataSource
String id = "db2adm";                  // Definir ID de usuario
Sring pw = "db2adm";                   // Definir contraseña
Connection con = ds.getConnection(id, pw);
                                          // Crear conexión
```

Como alternativa, si crea y despliega el objeto DataSource, puede establecer el ID de usuario y la contraseña invocando los métodos DataSource.setUser y DataSource.setPassword después de crear el objeto DataSource. Por ejemplo:

```
import java.sql.*;          // Base JDBC
import COM.ibm.db2.jdbc.*;                // Implementación de JDBC para
                                          // controlador JDBC de DB2 de tipo 2
...
DB2DataSource db2ds = new DB2DataSource();
                                            // Crear el objeto DataSource
db2ds.setDatabaseName("toronto");      // Definir la ubicación
db2ds.setUser("db2adm");               // Definir el ID de usuario
db2ds.setPassword("db2adm");           // Definir la contraseña
```

Capítulo 7. 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 el manualmente el mandato javac para crear aplicaciones JDBC.

Los pasos siguientes muestran cómo crear y ejecutar el applet de ejemplo `Applt.java` de JDBC.

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, 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 el manualmente el mandato javac para crear aplicaciones JDBC.

Los pasos siguientes muestran cómo crear y ejecutar la aplicación SQLJ de ejemplo `DbInfo` de JDBC.

1. Compile `DbInfo.java` para obtener el archivo `DbInfo.class` mediante este mandato:

   ```
   javac DbInfo.java
   ```

2. Si está ejecutando una aplicación Java sobre 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.

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
- 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.
  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.

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.

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, 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.

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.

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 sobre 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
   ```

4. Y finalmente, esto.

Este es un ejemplo... Insertar la lengüeta A en la ranura B.

Ahora, puede realizar lo siguiente...

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, 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).

"${nombreprog}.sqlj"
El archivo fuente de SQLJ. El mandato `progname=${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.

"${nombreprog}_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.

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.

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).

"${nombreprog}.sqlj"
El archivo fuente de SQLJ. El mandato `progname=${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.

"${nombreprog}_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 El conversor 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 8. Diagnóstico de problemas con IBM Data Server Driver para JDBC y SQLJ

Para obtener datos para diagnosticar problemas de SQLJ o JDBC mediante IBM Data Server Driver para JDBC y SQLJ, recoja datos de rastreo y ejecute programas de utilidad para formatear los datos de rastreo.

Debe ejecutar los programas de utilidad de rastreo y diagnóstico sólo bajo la dirección del soporte de software de IBM.

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 conectividad JDBC y SQLJ de tipo 4 o IBM Data Server Driver para conectividad JDBC y SQLJ 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.

Procedimiento 2:

1. Si utiliza la interfaz DataSource para conectar con una fuente de datos, invoque el método DB2BaseDataSource.setTraceLevel para definir el tipo de rastreo que necesite. El nivel de rastreo por omisión es `TRACE_ALL`. Consulte ″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 registro cronológico.

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 el 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.

Ejemplo de utilización de propiedades de configuración para iniciar un rastreo de JDBC

Puede controlar el rastreo de aplicaciones JDBC sin modificar esas aplicaciones.

Suponga que desea reunir datos de rastreo para un programa llamado Test.java, el cual utiliza el IBM Data Server Driver para conectividad JDBC y SQLJ 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 fichero 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 Test1.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 Test1.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 Test1.java, deberá emitir un mandato como el siguiente:

```
java -Ddb2.jcc.propertiesFile=/Test/jcc.properties Test1
```

Suponga que Test1.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 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 conectividad JDBC y SQLJ de tipo 4.

Figura 57. 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();
    }
  }
}
```

Capítulo 9. 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.

Actualmente, las bases de datos IBM Informix Dynamic Server no son compatibles con esta función.

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.

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 58. 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 del IBM Data Server Driver para JDBC y SQLJ

El IBM Data Server Driver para JDBC y SQLJ proporciona un recurso para controlar dinámicamente los rastreos del 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, tal 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 el IBM Data Server Driver para JDBC y SQLJ, y hacer que el agente JMX esté disponible para los clientes.

El controlador de rastreo remoto necesita Java Standard Edition, Versión 6 o posterior.

Siga estos pasos para habilitar el controlador de rastreo remoto:

1. Habilite JMX para el 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 autenticación 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://java.sun.com/javase/6/docs/technotes/guides/management/agent.html`

– Escriba un agente JMX. Esta técnica se describe también en:

`http://java.sun.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;

    ...
    // Autenticar 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.

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://java.sun.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:

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: 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 10. Java 2 Platform, Enterprise Edition

Java 2 Platform Enterprise Edition (J2EE) reduce el coste y la complejidad de desarrollar estos servicios de varios niveles, lo que da lugar a servicios que se pueden desplegar rápidamente y se pueden mejorar fácilmente según los requisitos de la empresa.

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 de 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 autenticación y autorización de Java

Servidor Java 2 Platform Enterprise Edition

Como elemento subyacente de un contenedor Java 2 Platform Enterprise Edition (J2EE) se encuentra el servidor del que forma parte el contenedor.

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 una base de datos a la que se pueda acceder a través de la API JDBC para el almacenamiento de los datos de la empresa.

Se puede acceder a la base de datos desde componentes de la web, Enterprise Beans y componentes cliente de la aplicación. No hace falta que se pueda acceder a la base de datos desde los applets.

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.

El IBM Data Server Driver para JDBC y SQLJ y el Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows aplican las especificaciones JTA y JTS.

Para el IBM Data Server Driver para conectividad JDBC y SQLJ 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 i5/OS.

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 J2EE 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.

El Controlador JDBC de DB2 de tipo 2 Driver proporciona estas dos clases de DataSource:
- COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource
- COM.ibm.db2.jdbc.DB2XADataSource

El 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 59 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 59. 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 60 en la página 212 muestra una aplicación que utiliza transacciones distribuidas. Mientras se ejecuta el código mostrado en el ejemplo, el servidor de aplicaciones también está ejecutando otros EJB que forman parte de la misma transacción distribuida. Cuando todos los EJB han invocado utx.commit(), el servidor de aplicaciones confirma la transacción distribuida completa. Si cualquiera de los EJB no se ejecuta satisfactoriamente, el servidor de aplicaciones retrotrae todo el trabajo hecho por todos los EJB que están asociados a la transacción distribuida.

```
javax.transaction.UserTransaction utx;
// Utilice el método begin sobre un objeto UserTransaction
// para indicar el inicio de una transacción distribuida.
utx.begin();
...
// Ejecute sentencias de SQL con un objeto Connection.
// No invoque los métodos commit ni rollback de Connection.
...
// Utilice el método commit sobre el objeto UserTransaction
// para hacer que se confirmen todas las ramas de transacción
// e indicar el final de la transacción distribuida.

utx.commit();
...
```

Figura 60. Ejemplo de transacción distribuida cuando se utiliza un servidor de aplicaciones

La Figura 61 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 61. 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();
      // Cree el objeto Xid de la transacción distribuida.
      // Este ejemplo utiliza la implementación com.ibm.db2.jcc.DB2Xid
      // de la interfaz Xid. Este Xid puede ser utilizado con cualquier
      // controlador JDBC que dé soporte a JTA.
      javax.transaction.xa.Xid xid1 =
                              new com.ibm.db2.jcc.DB2Xid(100, gtrid, bqual);

      // Inicie la transacción distribuida en las dos conexiones.
      // NO es necesario iniciar y finalizar las dos conexiones juntas.
      // Esto puede hacerse en hebras diferentes, junto con sus operaciones de SQL.
      xares1.start(xid1, javax.transaction.xa.XAResource.TMNOFLAGS);
      xares2.start(xid1, javax.transaction.xa.XAResource.TMNOFLAGS);
...
      // Ejecute las operaciones de SQL en la conexión 1.
      // Ejecute las operaciones de SQL en la conexión 2.
...
      // Ahora finalice la transacción distribuida en las dos conexiones.
      xares1.end(xid1, javax.transaction.xa.XAResource.TMSUCCESS);
      xares2.end(xid1, javax.transaction.xa.XAResource.TMSUCCESS);

      // Si el trabajo de la conexión 2 se ha realizado en otra hebra,
      // es necesaria aquí una llamada a thread.join() para esperar
      // a que termine el trabajo de la conexión 2.

      try
      { // Ahora prepare ambas ramas de la transacción distribuida.
        // Ambas ramas se deben preparar satisfactoriamente para
        // poder confirmar los cambios.
        // Si la transacción distribuida falla, se emite una
        // excepción XAException.
        rc1 = xares1.prepare(xid1);
        if(rc1 == javax.transaction.xa.XAResource.XA_OK)
        { // La preparación fue satisfactoria. Prepare la segunda conexión
          rc2 = xares2.prepare(xid1);
          if(rc2 == javax.transaction.xa.XAResource.XA_OK)
          { // Ambas conexiones se prepararon satisfactoriamente
            // y ninguna de ella era de solo lectura.
            xares1.commit(xid1, false);
            xares2.commit(xid1, false);
          }
          else if(rc2 == javax.transaction.xa.XAException.XA_RDONLY)
          { // La segunda conexión es de solo lectura, por lo que
            // solo se confirma la primera conexión.
            xares1.commit(xid1, false);
          }
        }
        else if(rc1 == javax.transaction.xa.XAException.XA_RDONLY)
        { // El SQL de la primera conexión es de solo lectura
          // (tal como un SELECT).
          // La preparación ha confirmado la conexión. Prepare la
          // segunda conexión.
          rc2 = xares2.prepare(xid1);
          if(rc2 == javax.transaction.xa.XAResource.XA_OK)
          { // La primera conexión es de solo lectura, pero la
            // segunda no lo es.
            // Confirme la segunda conexión.
            xares2.commit(xid1, false);
```

```
        }
        else if(rc2 == javax.transaction.xa.XAException.XA_RDONLY)
        { // Ambas conexiones son de solo lectura, y ambas
          // están ya confirmadas, por lo que no es necesaria
          // ninguna otra acción.
        }
      }
    }      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 que es el destino de transacciones distribuidas.

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 y IBM Data Server Driver para conectividad JDBC y SQLJ 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:

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 sondeo de conexiones JDBC y SQLJ

La *agrupación de conexiones* forma parte del soporte DataSource de JDBC y está soportada por el IBM Data Server Driver para JDBC y SQLJ.

El 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 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. Concentrador de conexiones de JDBC y equilibrado de carga Sysplex

Las aplicaciones Java que utilizan IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 4 para acceder a los servidores DB2 para z/OS pueden utilizar las funciones de concentrador de conexión y equilibrado de carga de trabajo Sysplex.

Las funciones de concentrador de conexiones y equilibrado de la carga de trabajo de Sysplex del IBM Data Server Driver para JDBC y SQLJ son similares a las funciones de concentrador de conexiones y equilibrado de la carga de trabajo de Sysplex de DB2 Connect.

La función del concentrador de conexiones del IBM Data Server Driver para JDBC y SQLJ puede reducir los recursos que las fuentes de datos DB2 para z/OS necesitan para trabajar con un número elevado de aplicaciones cliente. La función del concentrador de conexiones de IBM Data Server Driver para JDBC y SQLJ permite que muchos objetos de conexión utilicen la misma conexión física, lo que reduce el número total de conexiones físicas con la fuente de datos.

El equilibrado de la carga de trabajo de Sysplex del IBM Data Server Driver para JDBC y SQLJ puede mejorar la disponibilidad de un grupo de compartimiento de datos. Cuando el equilibrado de la carga de trabajo de Sysplex está habilitado, el controlador obtiene regularmente información de estado sobre los miembros de un grupo de compartimiento de datos. El controlador utiliza esta información para determinar el miembro de compartimiento de datos al que debe dirigirse la siguiente transacción. Con el equilibrado de la carga de trabajo de Sysplex, el servidor DB2 para z/OS y Workload Manager for z/OS (WLM) garantizan que el trabajo se distribuye de forma eficaz entre los miembros del grupo de compartimiento de datos y que el trabajo se transfiere a otro miembro del grupo de compartimiento de datos si uno de los miembros sufre una anomalía.

El IBM Data Server Driver para JDBC y SQLJ utiliza *objetos de transporte* y una *agrupación de objetos de transporte global* para poder utilizar el concentrador de conexiones y el equilibrado de la carga de trabajo de Sysplex. Existe un solo objeto de transporte para cada conexión física con la fuente de base de datos. Cuando habilita el concentrador de conexiones y el equilibrado de la carga de trabajo de Sysplex, define el número máximo de conexiones con la fuente de base de datos que pueden existir en cualquier momento; para ello establece el número máximo de objetos de transporte.

A nivel de controlador, establezca límites respecto al número de objetos de transporte utilizando propiedades de configuración del IBM Data Server Driver para JDBC y SQLJ.

A nivel de conexión, puede utilizar propiedades de DataSource para habilitar e inhabilitar el concentrador de conexiones de IBM Data Server Driver para JDBC y SQLJ y el equilibrado de la carga de trabajo de Sysplex, y definir límites respecto al número de objetos de transporte. Puede definir estas propiedades al obtener una conexión utilizando la interfaz DataSource o la interfaz DriverManager.

La agrupación de objetos de transporte global se puede supervisar de cualquiera de los modos siguientes:

- Mediante rastreos que se inician con las propiedades de configuración del IBM Data Server Driver para JDBC y SQLJ
- Mediante una interfaz de programación de aplicaciones

Ejemplo de habilitación del concentrador de conexiones de IBM Data Server Driver para JDBC y SQLJ

Para utilizar el concentrador de conexiones de IBM Data Server Driver para JDBC y SQLJ y las funciones de equilibrado de la carga de trabajo de Sysplex con WebSphere Application Server, es necesario que configure esas funciones.

Requisitos del servidor:
- WLM para z/OS
- DB2 UDB para OS/390 y z/OS Versión 7 o posterior, configurado para el compartimiento de datos

  El valor por omisión para registros especiales en todos los miembros del grupo de compartimiento de datos debe ser el mismo. La razón de ello se debe a que cuando el IBM Data Server Driver para JDBC y SQLJ equilibra las cargas en cada miembro del grupo de compartimiento de datos, mueve la conexión del usuario de un miembro a otro. Si el usuario ha configurado algún valor de registro especial en el miembro de compartimiento de datos original, el controlador restablecerá todos los registros especiales en sus valores por omisión y aplicará los cambios de registro especiales en el nuevo miembro. Sin embargo, el IBM Data Server Driver para JDBC y SQLJ no tiene forma de determinar los valores por omisión de todos los miembros. Si dos miembros tienen valores por omisión diferentes, el resultado de una sentencia de SQL puede ser diferente, en función del miembro en el que se ejecute la sentencia.

Requisitos del cliente:
- El IBM Data Server Driver para JDBC y SQLJ debe estar en el nivel de FixPak 10
- WebSphere Application Server, Versión 5.1 o posterior

El procedimiento siguiente es un ejemplo de habilitación de las funciones de equilibro de la carga de trabajo Sysplex y del concentrador de conexión del IBM Data Server Driver para JDBC y SQLJ con WebSphere Application Server. Los valores que se especifican no se dan como recomendación. Los valores deben determinarse basándose en factores como los que indicamos a continuación:
- Disponibilidad de los recursos del sistema
- El número de conexiones físicas disponibles
- La relación deseada de los objetos de conexión con los objetos de transporte

1. Compruebe que IBM Data Server Driver para JDBC y SQLJ está al nivel correcto para utilizar el concentrador de conexiones y el equilibrado de la carga de trabajo de Sysplex. Para ello siga estos pasos:
   a. Emita el mandato siguiente en el procesador 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 2.7 o posterior.
      ```
      [jcc] Driver: IBM Data Server Driver for JDBC and SQLJ Architecture nnn xxx
      ```
2. Establezca propiedades de configuración de IBM Data Server Driver para JDBC y SQLJ para habilitar el concentrador de conexiones o el equilibrado de la carga de trabajo de Sysplex 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 las propiedades de configuración siguientes:
   - db2.jcc.minTransportObjects
   - db2.jcc.maxTransportObjects
   - db2.jcc.maxTransportObjectWaitTime
   - db2.jcc.dumpPool
   - db2.jcc.dumpPoolStatisticsOnScheduleFile

   Empiece con una configuración similar a la siguiente:

```
db2.jcc.minTransportObjects=0
db2.jcc.maxTransportObjects=1500
db2.jcc.maxTransportObjectWaitTime=-1
db2.jcc.dumpPool=0
db2.jcc.dumpPoolStatisticsOnScheduleFile=/home/WAS/logs/srv1/poolstats
```

c. Añada la vía de acceso del directorio para DB2JccConfiguration.properties a la classpath del IBM Data Server Driver para JDBC y SQLJ de WebSphere Application Server.

3. Establezca las propiedades de la fuente de datos de IBM Data Server Driver para JDBC y SQLJ para habilitar el concentrador de conexiones o el equilibrado de la carga de trabajo de Sysplex:

   En la consola administrativa de WebSphere Application Server, establezca las propiedades siguientes para la fuente de datos que su aplicación utiliza para conectar con la fuente de datos:
   - enableSysplexWLB
   - enableConnectionConcentrator
   - maxTransportObjects

   Supongamos que desea la función del concentrador de conexión y la de equilibrado de carga de trabajo Sysplex. Empiece con una configuración similar a la siguiente:

Tabla 27. Ejemplo de configuración de las propiedades de una fuente de datos para el equilibrado de carga de trabajo Sysplex y el concentrador de conexión del IBM Data Server Driver para JDBC y SQLJ

Propiedad	Valor
enableSysplexWLB	true[1]
maxTransportObjects	100

Nota:

1. enableConnectionConcentrator está establecido en true por omisión ya que enableSysplexWLB está establecido en true.

4. Reinicie WebSphere Application Server.

Técnicas para supervisar el concentrador de conexiones de IBM Data Server Driver para JDBC y SQLJ y el equilibrado de la carga de trabajo de Sysplex

Para supervisar el concentrador de conexiones de IBM Data Server Driver para JDBC y SQLJ y el equilibrado de la carga de trabajo de Sysplex, es necesario 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 del 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 propiedad de configuración causa mensajes de error Sysplez y de agrupación de vuelco que deben grabarse 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 el 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 el 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 el 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 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 Dynamic Server (IDS).

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 28. 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	SMALLINT
boolean[1], byte[1], java.lang.Boolean	SMALLINT
int, java.lang.Integer	INTEGER
long, java.lang.Long	BIGINT
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
byte[]	CHAR(*n*) FOR BIT DATA[5]
byte[]	VARCHAR(*n*) FOR BIT DATA[7]
byte[]	BINARY(*n*)[5]
byte[]	VARBINARY(*n*)[7]

Tabla 28. 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[]	BLOB[9]
byte[]	ROWID
byte[]	XML
java.sql.Blob	BLOB
java.sql.Blob	XML
java.sql.Clob	CLOB
java.sql.Clob	DBCLOB[9]
java.sql.Clob	XML
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP
java.io.ByteArrayInputStream	BLOB
java.io.StringReader	CLOB
java.io.ByteArrayInputStream	CLOB
java.io.InputStream	XML
com.ibm.db2.jcc.DB2RowID(en desuso)	ROWID
java.sql.RowId	ROWID
com.ibm.db2.jcc.DB2Xml(en desuso)	XML
java.sql.SQLXML	XML

Notas:

1. El servidor de bases 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 para z/OS Versión 9.1 o posterior, o con servidores de bases de datos DB2 V9.5 para Linux, UNIX y Windows o posterior. 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.

Tipos de datos para recuperar datos de columnas de tabla

La tabla siguiente resume las correlaciones de tipos de datos DB2 o IDS 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 29. 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
BIGINT	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*)	byte[]	Ninguno
VARBINARY(*n*)	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]

Tabla 29. 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
DBCLOB(*m*)	No existe un equivalente exacto. Utilice java.sql.Clob.	
ROWID	java.sql.RowId	byte[], com.ibm.db2.jcc.DB2RowID (en desuso)
XML	java.sql.SQLXML	byte[], java.lang.String, java.io.InputStream, java.io.Reader
DATE	java.sql.Date	java.sql.String, java.sql.Timestamp
TIME	java.sql.Time	java.sql.String, java.sql.Timestamp
TIMESTAMP	java.sql.Timestamp	java.sql.String, java.sql.Date, java.sql.Time, java.sql.Timestamp

Notas:

1. *n*=16 o *n*=34.
2. DECFLOAT 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 V9.5 para Linux, UNIX y Windows o posterior. 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.

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 IDS 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 30. 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[1]	BIT	SMALLINT
byte[1]	TINYINT	SMALLINT
short, java.lang.Short	SMALLINT	SMALLINT
int, java.lang.Integer	INTEGER	INTEGER
long	BIGINT	BIGINT
float, java.lang.Float	REAL	REAL
float, java.lang.Float	FLOAT	REAL
double, java.lang.Double	DOUBLE	DOUBLE
java.math.BigDecimal	NUMERIC	DECIMAL

Tabla 30. 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.math.BigDecimal	DECIMAL	DECIMAL
java.lang.String	CHAR	CHAR
java.lang.String	CHAR	GRAPHIC
java.lang.String	VARCHAR	VARCHAR
java.lang.String	VARCHAR	VARGRAPHIC
java.lang.String	LONGVARCHAR	VARCHAR
java.lang.String	VARCHAR	CLOB
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
byte[]	VARBINARY	VARBINARY
byte[]	LONGVARBINARY	VARCHAR FOR BIT DATA
byte[]	VARBINARY	BLOB[3]
byte[]	LONGVARBINARY	BLOB[3]
java.sql.Date	DATE	DATE
java.sql.Time	TIME	TIME
java.sql.Timestamp	TIMESTAMP	TIMESTAMP
java.sql.Blob	BLOB	BLOB
java.sql.Clob	CLOB	CLOB
java.sql.Clob	CLOB	DBCLOB
java.io.ByteArrayInputStream	Ninguno	BLOB
java.io.StringReader	Ninguno	CLOB
java.io.ByteArrayInputStream	Ninguno	CLOB
com.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
java.sql.SQLXML	java.sql.Types.SQLXML	XML AS CLOB
java.sql.Array[4]	java.sql.Types.ARRAY	ARRAY

Tabla 30. 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

Notas:

1. Un procedimiento almacenado o función definida por el usuario que esté definido con un parámetro de tipo SMALLINT se puede invocar con un parámetro de tipo boolean o byte. Sin embargo, esto no es aconsejable.
2. 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 de las rutinas Java no se pueden utilizar para conexiones con para Linux, UNIX y Windows. 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. Los parámetros ARRAY se pueden utilizar solamente para procedimientos almacenados.

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 31. 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[1]
SMALLINT	short, java.lang.Integer
INTEGER	int, java.lang.Integer
BIGINT	long, java.lang.Long
REAL	float, java.lang.Float
DOUBLE	double, java.lang.Double
DECIMAL	java.math.BigDecimal
DECFLOAT[2]	java.math.BigDecimal
CHAR	java.lang.String
VARCHAR	java.lang.String
CHAR FOR BIT DATA	byte[]
VARCHAR FOR BIT DATA	byte[]
BINARY	byte[]
VARBINARY	byte[]

Tabla 31. 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 (continuación)

Tipo de datos de SQL en CREATE PROCEDURE o CREATE FUNCTION	Tipo de datos en procedimiento almacenado Java o función definida por el usuario[1]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp
BLOB	java.sql.Blob
CLOB	java.sql.Clob
DBCLOB	java.sql.Clob
ROWID	java.sql.Types.ROWID
XML AS CLOB	java.sql.Types.SQLXML

Notas:

1. 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.
2. 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 de las rutinas Java no se pueden utilizar para conexiones con para Linux, UNIX y Windows. La utilización de DECFLOAT requiere el SDK de Java Versión 5 (1.5) o posterior.

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" +
 "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:

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 conectividad JDBC y SQLJ 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.

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 conectividad JDBC y SQLJ de tipo 4 o IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 2.

Para IBM Data Server Driver para conectividad JDBC y SQLJ 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 IDS, *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 conectividad JDBC y SQLJ 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.

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 Dynamic Server (IDS), el valor por omisión depende del tipo de fuente de datos. La tabla siguiente muestra los valores por omisión.

Tabla 32. Niveles de aislamiento por omisión para bases de datos IDS

Tipo de fuente de datos	Nivel de aislamiento por omisión
Base de datos compatible con ANSI con registro cronológico	java.sql.Connection.TRANSACTION_SERIALIZABLE
Base de datos sin registro cronológico	java.sql.Connection.TRANSACTION_READ_UNCOMMITTED
Base de datos no compatible con ANSI con registro cronológico	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 boolean.

deferPrepares se puede utilizar para IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 2 con DB2 Database para Linux, UNIX y Windows, y IBM Data Server Driver para conectividad JDBC y SQLJ 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. Esto 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.

description
Descripción de la fuente de datos. El tipo de datos de esta propiedad es String.

downgradeHoldCursorsUnderXa
Especifica si se pueden abrir cursores retenidos en conexiones XA.

downgradeHoldCursorsUnderXa se aplica a:

- IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 4 con servidores DB2 para z/OS.
- IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 4 o IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 2 con servidores DB2 Database para Linux, UNIX y Windows.

El valor por omisión es `false`, que significa que no se pueden abrir cursores retenidos en conexiones XA.

Si se establece `true` como valor de downgradeHoldCursorsUnderXa, se podrán abrir cursores retenidos en conexiones XA. No obstante, los cursores en conexiones XA tienen las restricciones siguientes:

- Los cursores retenidos que se abren en una conexión XA no tienen comportamiento de WITH HOLD. Los cursores retenidos que se abren en una conexión XA se cierran al realizar la operación XA End.
- Los cursores abiertos antes de XA Start en una transacción local se cierran 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.

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:

- 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.

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 el IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 2 en DB2 para z/OS.

logWriter
Es la corriente de salida donde se escriben todos los mensajes de registro de anotaciones y de rastreo correspondientes al objeto DataSource. El tipo de datos de esta propiedad es java.io.PrinterWriter. El valor por omisión es nulo, que significa que no se generan datos de registro de anotaciones ni de rastreo para DataSource.

password
Es la contraseña que se debe utilizar para establecer conexiones. El tipo de datos de esta propiedad es String. Cuando utiliza la interfaz DataSource para establecer una conexión, puede alterar temporalmente el valor de esta propiedad invocando esta modalidad del método DataSource.getConnection:

```
getConnection(user, password);
```

portNumber
El número de puerto en que el servidor DRDA escucha peticiones. El tipo de datos de esta propiedad es int.

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)
Cerrar los cursores después de una operación de confirmación.

DB2BaseDataSource.CLOSE_CURSORS_AT_COMMIT (2)
Dejar los cursores abiertos después de una operación de confirmación.

DB2BaseDataSource.NOT_SET (0)
Cerrar los cursores después de una operación de confirmación. Esto es el valor por omisión.

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 es aplicable a las conexiones con IDS.

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 es aplicable a las conexiones con IDS.

PLUGIN_SECURITY (15)
Seguridad por plug-in. 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 es aplicable a las conexiones con IDS.

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 de securityMechanism es CLEAR_TEXT_PASSWORD_SECURITY. Si el servidor no admite CLEAR_TEXT_PASSWORD_SECURITY pero sí admite ENCRYPTED_USER_AND_PASSWORD_SECURITY, el 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.

sendDataAsIs
Especifica que el 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 boolean. 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.

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 el 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 boolean. El valor por omisión es `false`, que significa que se sobrescribe el archivo especificado por la propiedad traceFile.

traceLevel
Especifica qué se debe rastrear. El tipo de datos de esta propiedad es int.

Puede especificar uno o más de los valores de rastreo siguientes con la propiedad traceLevel:
- `com.ibm.db2.jcc.DB2BaseDataSource.TRACE_NONE` (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 conectividad JDBC y SQLJ 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
  ```

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 el IBM Data Server Driver para conectividad JDBC y SQLJ 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.

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 la puesta 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 la puesta en antememoria para un objeto PooledConnection ya existente.

maxStatements es aplicable solamente a IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 4.

Propiedades comunes de IBM Data Server Driver para JDBC y SQLJ para DB2 para z/OS y DB2 Database para Linux, UNIX y Windows

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:

clientRerouteAlternateServerName
Especifica uno o más nombres de servidores alternativos para el redireccionamiento del cliente. Se utiliza clientRerouteAlternateServerName cuando el redireccionamiento del cliente está configurado sin un almacén de datos JNDI. El tipo de datos de esta propiedad es String.

Si especifica más de un nombre de servidor, delimite los nombres con comas (,). 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 conectividad JDBC y SQLJ de tipo 2 con DB2 Database para Linux, UNIX y Windows y IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 4.

clientRerouteAlternatePortNumber
Especifica uno o más números de puerto alternativos para el redireccionamiento del cliente. Se utiliza clientRerouteAlternatePortNumber cuando el redireccionamiento del cliente está configurado sin un almacén de datos JNDI. El tipo de datos de esta propiedad es String.

Si especifica más de un número de puerto, delimite los números de puerto con comas (,). 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 conectividad JDBC y SQLJ de tipo 2 con DB2 Database para Linux, UNIX y Windows y IBM Data Server Driver para conectividad JDBC y SQLJ 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 conectividad JDBC y SQLJ 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 conectividad JDBC y SQLJ de tipo 4, y a las conexiones que se establecen mediante la interfaz DataSource.

Si no se define clientRerouteServerListJNDIContext, el 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)
```

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.

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 consultas materializadas.

NONE
Indica que no se tendrá en cuenta ninguna tabla de consulta materializada.

SYSTEM
Indica que sólo se tendrán en cuenta las tablas de consultas materializadas de renovación diferida mantenidas por el sistema.

USER
Indica que sólo se tendrán en cuenta las tablas de consultas materializadas de renovación diferida mantenidas por el usuario.

currentPackagePath
Especifica un 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 del IBM Data Server Driver para JDBC y SQLJ. Para seleccionar una instancia del 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 del 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.

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 de DB2 para i5/OS, los cuales no soportan cursores estáticos sensibles, el valor `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 valores decimales de coma flotante en servidores de bases 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 por defecto (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 más que la mitad (0,5) del valor del dígito en la posición izquierda siguiente, el coeficiente del resultado se incrementa en 1. Si representan menos que la mitad, el coeficiente del resultado no se ajusta (es decir, los dígitos descartados no se tienen en cuenta). 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. El IBM Data Server Driver para JDBC y SQLJ no utiliza la propiedad decimalRoundingMode para establecer la modalidad de redondeo en la fuente de datos.

El IBM Data Server Driver para JDBC y SQLJ utiliza los valores siguientes para establecer la modalidad de redondeo:

- Para servidores de bases de datos DB2 para z/OS o DB2 Database para Linux, UNIX y Windows, la modalidad de redondeo es ROUND_HALF_EVEN para valores decimales de coma flotante.

Si decimalRoundingMode está definido, se utiliza el valor de decimalRoundingMode para establecer el registro especial CURRENT DECFLOAT ROUNDING MODE en los servidores de bases de datos DB2 para z/OS.

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 boolean. 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 conectividad JDBC y SQLJ 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 conectividad JDBC y SQLJ de tipo 4.

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. El valor por omisión de maxRetriesForClientReroute es MAX_RETRIES_NOT_SET (-1).

Si maxRetriesForClientReroute o retryIntervalForClientReroute no está establecido, IBM Data Server Driver para JDBC y SQLJ realiza reintentos durante 10 minutos.

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, 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.

readOnly
Especifica si la conexión es de solo lectura. El tipo de datos de esta propiedad es boolean. 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, el 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)
Cerrar 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)
Dejar abiertos 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. Esto es el valor por omisión.

Los valores válidos son HOLD_CURSORS_OVER_COMMIT (1) o CLOSE_CURSORS_AT_COMMIT (2). Estos valores son los mismos que las constantes ResultSet.HOLD_CURSORS_OVER_COMMIT y ResultSet.CLOSE_CURSORS_AT_COMMIT que se definen en JDBC 3.0.

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. El valor por omisión de retryIntervalForClientReroute es RETRY_INTERVAL_NOT_SET (-1).

Si maxRetriesForClientReroute o retryIntervalForClientReroute no está establecido, IBM Data Server Driver para JDBC y SQLJ realiza reintentos durante 10 minutos.

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.

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.

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.

supportsAsynchronousXARollback
Especifica si el 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	*h:mm* am o *h: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`.

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.

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:

com.ibm.db2.jcc.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 4.0 y versiones posteriores, para las 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 para i5/OS V5R3 o posterior

Para las versiones anteriores de estos sistemas de bases de datos, son aplicables las reglas correspondientes al valor com.ibm.db2.jcc.DB2BaseDataSource.NO de useJDBC4ColumnNameAndLabelSemantics.

com.ibm.db2.jcc.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.

com.ibm.db2.jcc.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 para com.ibm.db2.jcc.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 com.ibm.db2.jcc.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 para i5/OS V5R3 o posterior
- Para las conexiones con versiones anteriores de esos sistemas de bases de datos, el comportamiento por omisión para useJDBC4ColumnNameAndLabelSemantics es com.ibm.db2.jcc.DB2BaseDataSource.NO.

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 la puesta 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 la puesta en antememoria para un objeto PooledConnection ya existente.

maxStatements es aplicable solamente a IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 4.

Propiedades comunes de IBM Data Server Driver para JDBC y SQLJ para DB2 para z/OS y IBM Informix Dynamic Server

Algunas de las propiedades de IBM Data Server Driver para JDBC y SQLJ son aplicables a servidores de bases de datos IDS y DB2 para z/OS.

Estas propiedades son las siguientes:

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 colocan sentencias dinámicas en la antememoria para una conexión si la propiedad no está establecida. Si la conexión está establecida con una fuente de datos IDS, se colocan 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 colocar 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.

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.

Propiedades comunes de IBM Data Server Driver para JDBC y SQLJ para servidores de bases de datos IBM Informix Dynamic Server y DB2 Database para Linux, UNIX y Windows

Algunas de las propiedades de IBM Data Server Driver para JDBC y SQLJ son aplicables a servidores de bases de datos IBM Informix Dynamic Server (IDS) y DB2 Database para Linux, UNIX y Windows.

Estas propiedades son las siguientes:

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.

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 32768 (32KB), son necesarios dos accesos al servidor de bases de datos para recuperar el conjunto de resultados. En cambio, si queryDataSize es igual a 61440 (60 KB), solamente es necesario acceder una 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 33. Valores mínimo, máximo y por omisión de queryDataSize

Fuente de datos	Valor mínimo de queryDataSize	Valor máximo de queryDataSize	Valor por omisión de queryDataSize
DB2 Database para Linux, UNIX y Windows	4096	65535	32767

Tabla 33. Valores mínimo, máximo y por omisión de queryDataSize (continuación)

Fuente de datos	Valor mínimo de queryDataSize	Valor máximo de queryDataSize	Valor por omisión de queryDataSize
IDS	4096	10485760	32767
DB2 para i5/OS	4096	65535	32767
DB2 para z/OS	No aplicable	No aplicable	El tamaño de los datos de la consulta es siempre 32767.

Propiedades de IBM Data Server Driver para JDBC y SQLJ correspondientes a 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 a la IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 4 con servidores DB2 Database para Linux, UNIX y Windows.

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, tales como 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 consultas materializadas en lugar de la 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.

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 boolean. 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 donde se utiliza Database Partitioning Feature (DPF).
- La clave de particionamiento permanece constante durante toda una transacción.

Si useTransactionRedirect es `true`, el 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 para z/OS 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", se crean registros contables en los puntos de confirmación. En otro caso, se crean registros contables al concluir la conexión física con la fuente de datos.

accountingInterval sólo es aplicable al IBM Data Server Driver para conectividad JDBC y SQLJ 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 al IBM Data Server Driver para conectividad JDBC y SQLJ 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, el 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 el 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.

clientAccountingInformation
Especifica información contable correspondiente al cliente actual de la conexión. Esta información se utiliza con fines de contabilidad de clientes. Este valor puede cambiar durante una conexión. El tipo de datos de esta propiedad es String. Para un servidor DB2 para z/OS, la longitud máxima es de 255 bytes. Una serie de caracteres vacía de 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 conectividad JDBC y SQLJ de tipo 2 sobre DB2 para z/OS.

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. Una serie de caracteres vacía de 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 conectividad JDBC y SQLJ de tipo 2 sobre DB2 para z/OS.

clientDebugInfo
Especifica un valor para el atributo de conexión CLIENT DEBUGINFO. Sirve para notificar al servidor DB2 para z/OS 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. CLIENT DEBUGINFO es utilizado por el depurador unificado de DB2. El tipo de datos de esta propiedad es String. La longitud máxima es 254 bytes.

Esta propiedad solamente es aplicable a IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 2 sobre DB2 para z/OS.

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.

Esta propiedad solamente es aplicable a IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 2 sobre DB2 para z/OS.

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`, el IBM Data Server Driver para JDBC y SQLJ proporcionará un valor db2jcc*nombre-hebra*.

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 conectividad JDBC y SQLJ 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 conectividad JDBC y SQLJ de tipo 2 sobre DB2 para z/OS.

currentSQLID
Especifica:

- El ID de autorización que se utiliza para la comprobación de autorizaciones en las sentencias CREATE, GRANT y REVOKE de SQL preparadas dinámicamente.
- El propietario de un espacio de tablas, base de datos, grupo de almacenamiento o sinónimo que es creado por una sentencia CREATE emitida dinámicamente.
- El calificador implícito de todos los nombres de tabla, vista, alias e índice especificados en sentencias de SQL dinámico.

La propiedad currentSQLID define el valor del registro especial CURRENT SQLID en un servidor DB2 para z/OS. Si la propiedad currentSQLID no está definida, el nombre de esquema por omisión es el valor del registro especial CURRENT SQLID.

enableConnectionConcentrator
Indica si la función del concentrador de conexión del IBM Data Server Driver para JDBC y SQLJ está habilitada. La función del concentrador de conexión está disponible sólo para conexiones con servidores DB2 para z/OS.

El tipo de datos de enableConnectionConcentrator es boolean. El valor por omisión es `false`. Sin embargo, si enableSysplexWLB se ha establecido en `true`, el valor por omisión será `true`.

enableSysplexWLB
Indica si la función de equilibrado de carga de trabajo Sysplex del IBM Data Server Driver para JDBC y SQLJ está habilitada. La función de equilibrado de carga de trabajo Sysplex está disponible sólo para conexiones con servidores DB2 para z/OS.

El tipo de datos de enableSysplexWLB es boolean. El valor por omisión es `false`. Si enableSysplexWLB se establece en `true`, enableConnectionConcentrator se establecerá en true por omisión.

jdbcCollection
Especifica el ID de colección de los paquetes que son utilizados por una instancia del 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 al IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 2 en DB2 para z/OS.

maxTransportObjects
Especifica el número máximo de objetos de transporte que se pueden utilizar para todas las conexiones con el objeto DataSource asociado. El 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 conexión 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 conexión 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.

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.

queryCloseImplicit
Especifica si los cursores se cierran inmediatamente después de que se hayan captado todas las filas. queryCloseImplicit se aplica sólo para el IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 4 con servidores de bases de datos DB2 para z/OS. Los valores posibles son `DB2BaseDataSource.QUERY_CLOSE_IMPLICIT_YES` (1) y `DB2BaseDataSource.QUERY_CLOSE_IMPLICIT_NO` (2). El valor por omisión es `DB2BaseDataSource.QUERY_CLOSE_IMPLICIT_YES`.

Un valor de `DB2BaseDataSource.QUERY_CLOSE_IMPLICIT_YES` puede tener un rendimiento mejor, ya que esta configuración reduce el tráfico de red.

sendCharInputsUTF8
Especifica si el 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 conectividad JDBC y SQLJ 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 el 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 el 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 conectividad JDBC y SQLJ 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 el 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.

sqljEnableClassLoaderSpecificProfiles
Especifica si el 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 boolean. El valor por omisión es `false`. sqljEnableClassLoaderSpecificProfiles es una propiedad de DataSource. Esta propiedad está pensada principalmente para utilizarla con WebSphere Application Server.

useRowsetCursor
Especifica si el IBM Data Server Driver para JDBC y SQLJ utiliza siempre operaciones FETCH de varias filas para cursores desplazables si la fuente de datos es compatible con esas operaciones. El tipo de datos de esta propiedad es boolean. El valor por omisión es `true`. La propiedad useRowsetCursor es aplicable solamente a conexiones con servidores de bases de datos DB2 para z/OS.

Las aplicaciones que utilizan la técnica 1 de JDBC para realizar operaciones de actualización o supresión de posición pueden seguir obteniendo los resultados deseados estableciendo useRowSetCursor en `false`. La técnica JDBC 1 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
```

Esas aplicaciones no operan debidamente si el IBM Data Server Driver para JDBC y SQLJ utiliza sentencias FETCH de varias filas. Establezca useRowSetCursor en `false` para asegurar que cada operación FETCH devuelva una sola fila.

Propiedades de IBM Data Server Driver para JDBC y SQLJ correspondientes a IBM Informix Dynamic Server

Algunas de las propiedades de IBM Data Server Driver para JDBC y SQLJ son aplicables solamente a bases de datos de IBM Informix Dynamic Server (IDS). Estas propiedades corresponden a variables de entorno de IDS.

Las propiedades específicas de IDS se deben especificar en letras mayúsculas. Los métodos getXXX y setXXX para propiedades específicas de IDS se forman anteponiendo get o set al nombre de la propiedad escrito en mayúsculas. Por ejemplo:

```
boolean dbDate = DB2BaseDateSource.getDBDATE();
```

Las propiedades específicas de IDS son:

DBANSIWARN
Especifica si IBM Data Server Driver para JDBC y SQLJ indica a la base de datos IDS si debe devolver un SQLWarning a la aplicación si una sentencia SQL utiliza una sintaxis que no sigue el estándar ANSI. El tipo de datos de esta propiedad es boolean. Los valores posibles son:

false ó 0
No envía un valor a la base de datos IDS que indica a la base de datos IDS si debe devolver un SQLWarning a la aplicación si una sentencia SQL utiliza una sintaxis que no sigue el estándar ANSI. Esto es el valor por omisión.

true ó 1
Envía un valor a la base de datos IDS que indica a la base de datos IDS que debe devolver un SQLWarning a la aplicación si una sentencia 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 IDS, sin embargo, no se puede utilizar la propiedad DBANSIWARN de IBM Data Server Driver para JDBC y SQLJ para restablecer la propiedad DBANSIWARN de IDS.

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 IDS
- 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.

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. Esto 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. Esto 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.

Propiedades de configuración del IBM Data Server Driver para JDBC y SQLJ

Las propiedades de configuración del IBM Data Server Driver para JDBC y SQLJ tienen un ámbito a nivel de controlador.

Puede establecer cualquiera de las propiedades de configuración del IBM Data Server Driver para JDBC y SQLJ siguientes. Todas las propiedades son opcionales.

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.

decimalRoundingMode
Especifica la modalidad de redondeo para valores decimales o valores decimales de coma flotante en servidores de bases de datos DB2 para z/OS o

DB2 Database para Linux, UNIX y Windows, y para valores decimales en todas las demás fuentes de datos que son compatibles con el tipo de datos decimal.

Los valores posibles son:

com.ibm.db2.jcc.DB2BaseDataSource.ROUND_UP (0)
Redondea el valor por exceso. Si todos los dígitos descartados son ceros, el resultado permanece inalterado salvo por la eliminación de los dígitos descartados. En otro caso, el coeficiente del resultado se incrementa en 1. Esta modalidad de redondeo no es recomendable al crear aplicaciones portables, pues no es compatible con el borrador de estándar del IEEE para aritmética de coma flotante.

com.ibm.db2.jcc.DB2BaseDataSource.ROUND_DOWN (1)
Redondea el valor por defecto (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 más que la mitad (0,5) del valor del dígito en la posición izquierda siguiente, el coeficiente del resultado se incrementa en 1. Si representan menos que la mitad, el coeficiente del resultado no se ajusta (es decir, los dígitos descartados no se tienen en cuenta). 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_HALF_DOWN (5)
Redondea el valor hasta el valor más cercano; si los valores son equidistantes, redondea el valor por defecto. Si los dígitos descartados representan un valor mayor 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. Esta modalidad de redondeo no es recomendable al crear aplicaciones portables, pues no es compatible con el borrador de estándar del IEEE para aritmética de coma flotante.

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. El IBM Data Server Driver para JDBC y SQLJ no utiliza la propiedad decimalRoundingMode para establecer la modalidad de redondeo en la fuente de datos.

El IBM Data Server Driver para JDBC y SQLJ utiliza los valores siguientes para establecer la modalidad de redondeo:

- Si la fuente de datos es DB2 para z/OS o DB2 Database para Linux, UNIX y Windows, la modalidad de redondeo es ROUND_HALF_EVEN para valores decimales o decimales de coma flotante.
- Para las demás fuentes de datos, la modalidad de redondeo es ROUND_DOWN para valores decimales.

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.

Un valor igual a

db2.jcc.traceDirectory o db2.jcc.override.traceDirectory
Habilita el rastreo del 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 del IBM Data Server Driver para JDBC y SQLJ que se guardará en 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, de forma que se especificaría:

```
db2.jcc.override.traceLevel=65
```

db2.jcc.sqljUncustomizedWarningOrException
Especifica la acción que el 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 El IBM Data Server Driver para JDBC y SQLJ no emitirá un aviso o una excepción cuando se ejecute una aplicación SQLJ no personalizada. Éste es el valor por omisión.

1 El IBM Data Server Driver para JDBC y SQLJ emitirá un aviso cuando se ejecute una aplicación SQLJ no personalizada.

2 El IBM Data Server Driver para JDBC y SQLJ emitirá una excepción cuando se ejecute una aplicación SQLJ no personalizada.

db2.jcc.traceFile o db2.jcc.override.traceFile
Habilita el rastreo del 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 del IBM Data Server Driver para JDBC y SQLJ que se guardará en 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. El tipo de datos de esta propiedad es boolean. El valor por omisión es `false`, que significa que se sobrescribe el archivo especificado por la propiedad traceFile.

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.

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 Dynamic Server (IDS) 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 34. Controladores JDBC para sistemas de bases de datos DB2 e IDS

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 Dynamic Server (IDS)
Controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows (en desuso)	DB2 Database para Linux, UNIX y Windows
Controlador JDBC de IBM Informix (Controlador JDBC de IDS)	IDS

Si un método tiene formatos de JDBC 2.0 y JDBC 3.0, el 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 35. Soporte para métodos Array

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ1 en la página 268	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
free[2]	Sí	No	No
getArray	Sí	No	Sí
getBaseType	Sí	No	Sí

Tabla 35. Soporte para métodos Array (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ1	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
getBaseTypeName	Sí	No	Sí
getResultSet	Sí	No	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 36. Soporte para métodos BatchUpdateException

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
Métodos heredados de java.lang.Exception	Sí	Sí	Sí
getUpdateCounts	Sí	Sí	Sí

Tabla 37. Soporte para métodos Blob

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
free[1]	Sí	No	No
getBinaryStream	Sí[2]	Sí	Sí
getBytes	Sí	Sí	Sí
longitud	Sí	Sí	Sí
posición	Sí	Sí	Sí
setBinaryStream[3]	Sí	No	No
setBytes[3]	Sí	No	No
truncate[3]	Sí	No	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. Este método no se puede utilizar en los casos siguientes:
   - Si la propiedad `fullyMaterializeLobData` está establecida en false.
   - Para un Blob que se pasa a un procedimiento almacenado Java.

Tabla 38. Soporte para métodos CallableStatement

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
Métodos heredados de java.sql.Statement	Sí	Sí	Sí
Métodos heredados de java.sql.PreparedStatement	Sí[1]	Sí	Sí
getArray	No	No	No
getBigDecimal	Sí[3]	Sí	Sí
getBlob	Sí[3]	Sí	Sí
getBoolean	Sí[3]	Sí	Sí
getByte	Sí[3]	Sí	Sí
getBytes	Sí[3]	Sí	Sí
getClob	Sí[3]	Sí	Sí
getDate	Sí[3,4]	Sí[4]	Sí
getDouble	Sí[3]	Sí	Sí
getFloat	Sí[3]	Sí	Sí
getInt	Sí[3]	Sí	Sí
getLong	Sí[3]	Sí	Sí
getObject	Sí[3,5]	Sí[5]	Sí
getRef	No	No	No
getRowId[2]	Sí	No	No
getShort	Sí[3]	Sí	Sí
getString	Sí[3]	Sí	Sí
getTime	Sí[3,4]	Sí[4]	Sí
getTimestamp	Sí[3,4]	Sí[4]	Sí
getURL	Sí	No	No
registerOutParameter	Sí[6]	Sí[6]	Sí[6]
setAsciiStream	Sí[7]	No	Sí
setBigDecimal	Sí[7]	No	Sí
setBinaryStream	Sí[7]	No	Sí
setBoolean	Sí[7]	No	Sí
setByte	Sí[7]	No	Sí
setBytes	Sí[7]	No	Sí
setCharacterStream	Sí[7]	No	Sí
setDate	Sí[7]	No	Sí
setDouble	Sí[7]	No	Sí
setFloat	Sí[7]	No	Sí
setInt	Sí[7]	No	Sí
setLong	Sí[7]	No	Sí
setNull	Sí[7,8]	No	Sí

Tabla 38. Soporte para métodos CallableStatement (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
setObject	Sí[7,]	No	Sí
setShort	Sí[7]	No	Sí
setString	Sí[7]	No	Sí
setTime	Sí[7]	No	Sí
setTimestamp	Sí[7]	No	Sí
setURL	Sí	No	No
wasNull	Sí	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. 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.
5. El formato siguiente del método getObject no se soporta:

   `getObject(int` *`Índiceparámetros`*`, java.util.Map` *`map`*`)`
6. El formato siguiente del método registerOutParameter no se soporta:

   `registerOutParameter(int` *`índiceParámetro`*`, int` *`tipoJdbc`*`, String` *`nombreTipo`*`)`
7. No permitido si la fuente de datos es DB2 para z/OS.
8. El formato siguiente de setNull no se soporta:

   `setNull(int` *`Índiceparámetros`*`, int` *`TipoJdbc`*`, String` *`nombreTipo`*`)`

Tabla 39. Soporte para métodos Clob

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
free[1]	Sí	No	No
getAsciiStream	Sí	Sí	Sí
getCharacterStream	Sí2 en la página 271	Sí	Sí
getSubString	Sí	Sí	Sí
longitud	Sí	Sí	Sí
posición	Sí	Sí	Sí
setAsciiStream[3]	Sí	No	Sí
setCharacterStream[3]	Sí	No	Sí
setString[3]	Sí	No	Sí
truncate[3]	Sí	No	Sí

Tabla 39. Soporte para métodos Clob (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS

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. Este método no se puede utilizar en los casos siguientes:
   - Si la propiedad `fullyMaterializeLobData` está establecida en false.
   - Para un Clob que se pasa a un procedimiento almacenado Java.

Tabla 40. Soporte para métodos Connection

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
clearWarnings	Sí	Sí	Sí
close	Sí	Sí	Sí
commit	Sí	Sí	Sí
createStatement	Sí	Sí[2]	Sí
createBlob[1]	Sí	No	No
createClob[1]	Sí	No	No
getAutoCommit	Sí	Sí	Sí
getCatalog	Sí	Sí	Sí
getClientInfo[3]	Sí	No	No
getHoldability	Sí	No	No
getMetaData	Sí	Sí	Sí
getTransactionIsolation	Sí	Sí	Sí
getTypeMap	No	No	Sí
getWarnings	Sí	Sí	Sí
isClosed	Sí	Sí	Sí
isReadOnly	Sí	Sí	Sí
isValid[3]	Sí	No	No
nativeSQL	Sí	Sí	Sí
prepareCall	Sí[4]	Sí	Sí
prepareStatement	Sí	Sí[2]	Sí
releaseSavepoint	Sí	No	No
rollback	Sí	Sí[2]	Sí
setAutoCommit	Sí	Sí	Sí
setCatalog	Sí	Sí	No
setClientInfo[3]	Sí	No	No
setReadOnly	Sí[5]	Sí	No
setSavepoint	Sí	No	Sí

Tabla 40. Soporte para métodos Connection (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
setTransactionIsolation	Sí	Sí	Sí
setTypeMap	No	No	Sí

Notas:

1. Esto es un método de JDBC 4.0.
2. El controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows no es compatible con los formatos JDBC 3.0 de este método.
3. Esto es un método de JDBC 4.0.
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 el 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 41. Soporte para métodos ConnectionEvent

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
Métodos heredados de java.util.EventObject	Sí	Sí	Sí
getSQLException	Sí	Sí	Sí

Tabla 42. Soporte para métodos ConnectionEventListener

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
connectionClosed	Sí	Sí	Sí
connectionErrorOccurred	Sí	Sí	Sí

Tabla 43. Soporte para métodos ConnectionPoolDataSource

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
getLoginTimeout	Sí	Sí	Sí
getLogWriter	Sí	Sí	Sí
getPooledConnection	Sí	Sí	Sí
setLoginTimeout	Sí[1]	Sí	Sí
setLogWriter	Sí	Sí	Sí

Nota:

1. Este método no está soportado para el IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 2 en DB2 para z/OS.

Tabla 44. Soporte para métodos DatabaseMetaData

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
allProceduresAreCallable	Sí	Sí	Sí
allTablesAreSelectable	Sí[1]	Sí	Sí[1]
dataDefinitionCausesTransactionCommit	Sí	Sí	Sí
dataDefinitionIgnoredInTransactions	Sí	Sí	Sí
deletesAreDetected	Sí	Sí	Sí
doesMaxRowSizeIncludeBlobs	Sí	Sí	Sí
getAttributes	Sí[2]	No	No
getBestRowIdentifier	Sí	Sí	Sí
getCatalogs	Sí	Sí	Sí
getCatalogSeparator	Sí	Sí	Sí
getCatalogTerm	Sí	Sí	Sí
getClientInfoProperties[6]	Sí	No	No
getColumnPrivileges	Sí	Sí	Sí
getColumns	Sí[7]	Sí[10]	Sí[10]
getConnection	Sí	Sí	Sí
getCrossReference	Sí	Sí	Sí
getDatabaseMajorVersion	Sí	No	No
getDatabaseMinorVersion	Sí	No	No
getDatabaseProductName	Sí	Sí	Sí
getDatabaseProductVersion	Sí	Sí	Sí
getDefaultTransactionIsolation	Sí	Sí	Sí
getDriverMajorVersion	Sí	Sí	Sí
getDriverMinorVersion	Sí	Sí	Sí
getDriverName	Sí[8]	Sí	Sí
getDriverVersion	Sí	Sí	Sí
getExportedKeys	Sí	Sí	Sí
getFunctionColumns[6]	Sí	No	No
getFunctions[6]	Sí	No	No
getExtraNameCharacters	Sí	Sí	Sí
getIdentifierQuoteString	Sí	Sí	Sí
getImportedKeys	Sí	Sí	Sí
getIndexInfo	Sí	Sí	Sí
getJDBCMajorVersion	Sí	No	No
getJDBCMinorVersion	Sí	No	No
getMaxBinaryLiteralLength	Sí	Sí	Sí
getMaxCatalogNameLength	Sí	Sí	Sí

Tabla 44. Soporte para métodos DatabaseMetaData (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
getMaxCharLiteralLength	Sí	Sí	Sí
getMaxColumnNameLength	Sí	Sí	Sí
getMaxColumnsInGroupBy	Sí	Sí	Sí
getMaxColumnsInIndex	Sí	Sí	Sí
getMaxColumnsInOrderBy	Sí	Sí	Sí
getMaxColumnsInSelect	Sí	Sí	Sí
getMaxColumnsInTable	Sí	Sí	Sí
getMaxConnections	Sí	Sí	Sí
getMaxCursorNameLength	Sí	Sí	Sí
getMaxIndexLength	Sí	Sí	Sí
getMaxProcedureNameLength	Sí	Sí	Sí
getMaxRowSize	Sí	Sí	Sí
getMaxSchemaNameLength	Sí	Sí	Sí
getMaxStatementLength	Sí	Sí	Sí
getMaxStatements	Sí	Sí	Sí
getMaxTableNameLength	Sí	Sí	Sí
getMaxTablesInSelect	Sí	Sí	Sí
getMaxUserNameLength	Sí	Sí	Sí
getNumericFunctions	Sí	Sí	Sí
getPrimaryKeys	Sí	Sí	Sí
getProcedureColumns	Sí7 en la página 278	Sí	Sí
getProcedures	Sí7 en la página 278	Sí	Sí
getProcedureTerm	Sí	Sí	Sí
getResultSetHoldability	Sí	No	No
getRowIdLifetime[6]	Sí	No	No
getSchemas	Sí9 en la página 278	Sí[10]	Sí[10]
getSchemaTerm	Sí	Sí	Sí
getSearchStringEscape	Sí	Sí	Sí
getSQLKeywords	Sí	Sí	Sí
getSQLStateType	Sí	No	No
getStringFunctions	Sí	Sí	Sí
getSuperTables	Sí[2]	No	No
getSuperTypes	Sí[2]	No	No
getSystemFunctions	Sí	Sí	Sí

Tabla 44. Soporte para métodos DatabaseMetaData (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
getTablePrivileges	Sí	Sí	Sí
getTables	Sí	Sí[10]	Sí[10]
getTableTypes	Sí	Sí	Sí
getTimeDateFunctions	Sí	Sí	Sí
getTypeInfo	Sí	Sí	Sí
getUDTs	No	Sí[11]	Sí[11]
getURL	Sí	Sí	Sí
getUserName	Sí	Sí	Sí
getVersionColumns	Sí	Sí	Sí
insertsAreDetected	Sí	Sí	Sí
isCatalogAtStart	Sí	Sí	Sí
isReadOnly	Sí	Sí	Sí
locatorsUpdateCopy	Sí[3]	Sí	Sí[3]
nullPlusNonNullIsNull	Sí	Sí	Sí
nullsAreSortedAtEnd	Sí[4]	Sí	Sí[4]
nullsAreSortedAtStart	Sí	Sí	Sí
nullsAreSortedHigh	Sí[5]	Sí	Sí[5]
nullsAreSortedLow	Sí[1]	Sí	Sí[1]
othersDeletesAreVisible	Sí	Sí	Sí
othersInsertsAreVisible	Sí	Sí	Sí
othersUpdatesAreVisible	Sí	Sí	Sí
ownDeletesAreVisible	Sí	Sí	Sí
ownInsertsAreVisible	Sí	Sí	Sí
ownUpdatesAreVisible	Sí	Sí	Sí
storesLowerCaseIdentifiers	Sí[1]	Sí	Sí[1]
storesLowerCaseQuotedIdentifiers	Sí[4]	Sí	Sí[4]
storesMixedCaseIdentifiers	Sí	Sí	Sí
storesMixedCaseQuotedIdentifiers	Sí	Sí	Sí
storesUpperCaseIdentifiers	Sí[5]	Sí	Sí[5]
storesUpperCaseQuotedIdentifiers	Sí	Sí	Sí
supportsAlterTableWithAddColumn	Sí	Sí	Sí
supportsAlterTableWithDropColumn	Sí[1]	Sí	Sí[1]
supportsANSI92EntryLevelSQL	Sí	Sí	Sí
supportsANSI92FullSQL	Sí	Sí	Sí
supportsANSI92IntermediateSQL	Sí	Sí	Sí
supportsBatchUpdates	Sí	Sí	Sí

Tabla 44. Soporte para métodos DatabaseMetaData (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
supportsCatalogsInDataManipulation	Sí[1]	Sí	Sí[1]
supportsCatalogsInIndexDefinitions	Sí	Sí	Sí
supportsCatalogsInPrivilegeDefinitions	Sí	Sí	Sí
supportsCatalogsInProcedureCalls	Sí[1]	Sí	Sí[1]
supportsCatalogsInTableDefinitions	Sí	Sí	Sí
SupportsColumnAliasing	Sí	Sí	Sí
supportsConvert	Sí	Sí	Sí
supportsCoreSQLGrammar	Sí	Sí	Sí
supportsCorrelatedSubqueries	Sí	Sí	Sí
supportsDataDefinitionAndDataManipulationTransactions	Sí	Sí	Sí
supportsDataManipulationTransactionsOnly	Sí	Sí	Sí
supportsDifferentTableCorrelationNames	Sí[4]	Sí	Sí[4]
supportsExpressionsInOrderBy	Sí	Sí	Sí
supportsExtendedSQLGrammar	Sí	Sí	Sí
supportsFullOuterJoins	Sí[3]	Sí	Sí[3]
supportsGetGeneratedKeys	Sí	No	No
supportsGroupBy	Sí	Sí	Sí
supportsGroupByBeyondSelect	Sí	Sí	Sí
supportsGroupByUnrelated	Sí	Sí	Sí
supportsIntegrityEnhancementFacility	Sí	Sí	Sí
supportsLikeEscapeClause	Sí	Sí	Sí
supportsLimitedOuterJoins	Sí	Sí	Sí
supportsMinimumSQLGrammar	Sí	Sí	Sí
supportsMixedCaseIdentifiers	Sí	Sí	Sí
supportsMixedCaseQuotedIdentifiers	Sí[3]	Sí	Sí[3]
supportsMultipleOpenResults	Sí[5]	No	Sí[5]
supportsMultipleResultSets	Sí[5]	Sí	Sí[5]
supportsMultipleTransactions	Sí	Sí	Sí
supportsNamedParameters	Sí	No	No
supportsNonNullableColumns	Sí	Sí	Sí
supportsOpenCursorsAcrossCommit	Sí[3]	Sí	Sí[3]
supportsOpenCursorsAcrossRollback	Sí	Sí	Sí
supportsOpenStatementsAcrossCommit	Sí[3]	Sí	Sí[3]
supportsOpenStatementsAcrossRollback	Sí[3]	Sí	Sí[3]
supportsOrderByUnrelated	Sí	Sí	Sí
supportsOuterJoins	Sí	Sí	Sí

Tabla 44. Soporte para métodos DatabaseMetaData (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
supportsPositionedDelete	Sí	Sí	Sí
supportsPositionedUpdate	Sí	Sí	Sí
supportsResultSetConcurrency	Sí	Sí	Sí
supportsResultSetHoldability	Sí	No	No
supportsResultSetType	Sí	Sí	Sí
supportsSavepoints	Sí[5]	No	Sí[5]
supportsSchemasInDataManipulation	Sí	Sí	Sí
supportsSchemasInIndexDefinitions	Sí	Sí	Sí
supportsSchemasInPrivilegeDefinitions	Sí	Sí	Sí
supportsSchemasInProcedureCalls	Sí	Sí	Sí
supportsSchemasInTableDefinitions	Sí	Sí	Sí
supportsSelectForUpdate	Sí	Sí	Sí
supportsStoredProcedures	Sí	Sí	Sí
supportsSubqueriesInComparisons	Sí	Sí	Sí
supportsSubqueriesInExists	Sí	Sí	Sí
supportsSubqueriesInIns	Sí	Sí	Sí
supportsSubqueriesInQuantifieds	Sí	Sí	Sí
supportsSuperTables	Sí	No	No
supportsSuperTypes	Sí	No	No
supportsTableCorrelationNames	Sí	Sí	Sí
supportsTransactionIsolationLevel	Sí	Sí	Sí
supportsTransactions	Sí	Sí	Sí
supportsUnion	Sí	Sí	Sí
supportsUnionAll	Sí	Sí	Sí
updatesAreDetected	Sí	Sí	Sí
usesLocalFilePerTable	Sí	Sí	Sí
usesLocalFiles	Sí	Sí	Sí

Tabla 44. Soporte para métodos DatabaseMetaData (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS

Notas:

1. Las fuentes de datos DB2 devuelven `false` para este método. Las fuentes de datos IDS devuelven `true`.
2. Este método solamente se puede utilizar para conexiones con DB2 Database para Linux, UNIX y Windows e IDS.
3. Cuando se utiliza IBM Data Server Driver para JDBC y SQLJ, las fuentes de datos DB2 y las fuentes de datos IDS devuelven `true` para este método. Cuando se utiliza el controlador JDBC de IDS, las fuentes de datos IDS devuelven `false`.
4. Cuando se utiliza IBM Data Server Driver para JDBC y SQLJ, las fuentes de datos DB2 y las fuentes de datos IDS devuelven `false` para este método. Cuando se utiliza el controlador JDBC de IDS, las fuentes de datos IDS devuelven `true`.
5. Las fuentes de datos DB2 devuelven `true` para este método. Las fuentes de datos IDS devuelven `false`.
6. Esto es un método de JDBC 4.0.
7. Este método devuelve la columna adicional descrita por la especificación JDBC 4.0.
8. 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.″
9. Se pueden utilizar el formato JDBC 4.0 y formatos anteriores de este método.
10. 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.
11. El método se puede ejecutar, pero devuelve un ResultSet vacío.

Tabla 45. Soporte para métodos DataSource

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
getConnection	Sí	Sí	Sí
getLoginTimeout	Sí	Sí[1]	Sí
getLogWriter	Sí	Sí	Sí
setLoginTimeout	Sí[2]	Sí[1]	Sí
setLogWriter	Sí	Sí	Sí

Notas:

1. El controlador JDBC de DB2 de tipo 2 no utiliza este valor.
2. Este método no está soportado para el IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 2 en DB2 para z/OS.

Tabla 46. Soporte para métodos DataTruncation

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
Métodos heredados de java.lang.Throwable	Sí	Sí	Sí
Métodos heredados de java.sql.SQLException	Sí	Sí	Sí
Métodos heredados de java.sql.SQLWarning	Sí	Sí	Sí
getDataSize	Sí	Sí	Sí
getIndex	Sí	Sí	Sí
getParameter	Sí	Sí	Sí
getRead	Sí	Sí	Sí
getTransferSize	Sí	Sí	Sí

Tabla 47. Soporte para métodos Driver

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
acceptsURL	Sí	Sí	Sí
connect	Sí	Sí	Sí
getMajorVersion	Sí	Sí	Sí
getMinorVersion	Sí	Sí	Sí
getPropertyInfo	Sí	Sí	Sí
jdbcCompliant	Sí	Sí	Sí

Tabla 48. Soporte para métodos DriverManager

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
deregisterDriver	Sí	Sí	Sí
getConnection	Sí	Sí	Sí
getDriver	Sí	Sí	Sí
getDrivers	Sí	Sí	Sí
getLoginTimeout	Sí	Sí[1]	Sí[1]
getLogStream	Sí	Sí	Sí
getLogWriter	Sí	Sí	Sí
println	Sí	Sí	Sí
registerDriver	Sí	Sí	Sí
setLoginTimeout	Sí[2]	Sí[1]	Sí[1]
setLogStream	Sí	Sí	Sí
setLogWriter	Sí	Sí	Sí

Tabla 48. Soporte para métodos DriverManager (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS

Notas:

1. El controlador JDBC de DB2 de tipo 2 no utiliza este valor.
2. Este método no está soportado para el IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 2 en DB2 para z/OS.

Tabla 49. Soporte para métodos ParameterMetaData

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
getParameterClassName	No	No	No
getParameterCount	Sí	No	No
getParameterMode	Sí	No	No
getParameterType	Sí	No	No
getParameterTypeName	Sí	No	No
getPrecision	Sí	No	No
getScale	Sí	No	No
isNullable	Sí	No	No
isSigned	Sí	No	No

Tabla 50. Soporte para métodos PooledConnection

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
addConnectionEventListener	Sí	Sí	Sí
addStatementEventListener[1]	Sí	No	No
close	Sí	Sí	Sí
getConnection	Sí	Sí	Sí
removeConnectionEventListener	Sí	Sí	Sí
removeStatementEventListener[1]	Sí	No	No

Notas:

1. Esto es un método de JDBC 4.0.

Tabla 51. Soporte para métodos PreparedStatement

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
Métodos heredados de java.sql.Statement	Sí	Sí	Sí
addBatch	Sí	Sí	Sí

Tabla 51. Soporte para métodos PreparedStatement (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
clearParameters	Sí	Sí	Sí
execute	Sí	Sí	Sí
executeQuery	Sí	Sí	Sí
executeUpdate	Sí	Sí	Sí
getMetaData	Sí	Sí	Sí
getParameterMetaData	Sí	Sí	Sí
setArray	No	No	No
setAsciiStream	Sí[1,2]	Sí	Sí
setBigDecimal	Sí	Sí	Sí
setBinaryStream	Sí[1,3]	Sí	Sí
setBlob	Sí[4]	Sí	Sí
setBoolean	Sí	Sí	Sí
setByte	Sí	Sí	Sí
setBytes	Sí	Sí	Sí
setCharacterStream	Sí[1,5]	Sí	Sí
setClob	Sí[6]	Sí	Sí
setDate	Sí[8]	Sí[8]	Sí[8]
setDouble	Sí	Sí	Sí
setFloat	Sí	Sí	Sí
setInt	Sí	Sí	Sí
setLong	Sí	Sí	Sí
setNull	Sí[9]	Sí[9]	Sí[9]
setObject	Sí	Sí	Sí
setRef	No	No	No
setRowId[7]	Sí	No	No
setShort	Sí	Sí	Sí
setString	Sí[10]	Sí[10]	Sí[10]
setTime	Sí[8]	Sí[8]	Sí[8]
setTimestamp	Sí[8]	Sí[8]	Sí[8]
setUnicodeStream	Sí	Sí	Sí
setURL	Sí	Sí	Sí

Tabla 51. Soporte para métodos PreparedStatement (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS

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 52. Soporte para métodos Ref

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
get BaseTypeName	No	No	No

Tabla 53. Soporte para métodos ResultSet

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
absolute	Sí	Sí	Sí
afterLast	Sí	Sí	Sí
beforeFirst	Sí	Sí	Sí
cancelRowUpdates	Sí	No	No
clearWarnings	Sí	Sí	Sí
close	Sí	Sí	Sí
deleteRow	Sí	No	No
findColumn	Sí	Sí	Sí

Tabla 53. Soporte para métodos ResultSet (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
first	Sí	Sí	Sí
getArray	No	No	No
getAsciiStream	Sí	Sí	Sí
getBigDecimal	Sí	Sí	Sí
getBinaryStream	Sí[1]	Sí	Sí
getBlob	Sí	Sí	Sí
getBoolean	Sí	Sí	Sí
getByte	Sí	Sí	Sí
getBytes	Sí	Sí	Sí
getCharacterStream	Sí	Sí	Sí
getClob	Sí	Sí	Sí
getConcurrency	Sí	Sí	Sí
getCursorName	Sí	Sí	Sí
getDate	Sí[2]	Sí[2]	Sí[2]
getDouble	Sí	Sí	Sí
getFetchDirection	Sí	Sí	Sí
getFetchSize	Sí	Sí	Sí
getFloat	Sí	Sí	Sí
getInt	Sí	Sí	Sí
getLong	Sí	Sí	Sí
getMetaData	Sí	Sí	Sí
getObject	Sí[3]	Sí[3]	Sí[3]
getRef	No	No	No
getRow	Sí	Sí	Sí
getRowId[9]	Sí	No	No
getShort	Sí	Sí	Sí
getStatement	Sí	Sí	Sí
getString	Sí	Sí	Sí
getTime	Sí[2]	Sí[2]	Sí[2]
getTimestamp	Sí[2]	Sí[2]	Sí[2]
getType	Sí	Sí	Sí
getUnicodeStream	Sí	Sí	Sí
getURL	Sí	Sí	Sí
getWarnings	Sí	Sí	Sí
insertRow	Sí	No	No
isAfterLast	Sí	Sí	Sí
isBeforeFirst	Sí	Sí	Sí
isFirst	Sí	Sí	Sí

Tabla 53. Soporte para métodos ResultSet (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
isLast	Sí	Sí	Sí
last	Sí	Sí	Sí
moveToCurrentRow	Sí	No	No
moveToInsertRow	Sí	No	No
next	Sí	Sí	Sí
previous	Sí	Sí	Sí
refreshRow	Sí	No	No
relative	Sí	Sí	Sí
rowDeleted	Sí	No	No
rowInserted	Sí	No	No
rowUpdated	Sí	No	No
setFetchDirection	Sí	Sí	Sí
setFetchSize	Sí	Sí	Sí
updateArray	No	No	No
updateAsciiStream	Sí[4]	No	No
updateBigDecimal	Sí	No	No
updateBinaryStream	Sí[5]	No	No
updateBlob	Sí[6]	No	No
updateBoolean	Sí	No	No
updateByte	Sí	No	No
updateBytes	Sí	No	No
updateCharacterStream	Sí[7]	No	No
updateClob	Sí[8]	No	No
updateDate	Sí	No	No
updateDouble	Sí	No	No
updateFloat	Sí	No	No
updateInt	Sí	No	No
updateLong	Sí	No	No
updateNull	Sí	No	No
updateObject	Sí	No	No
updateRef	No	No	No
updateRow	Sí	No	No
updateRowId[9]	Sí	No	No
updateShort	Sí	No	No
updateString	Sí	No	No
updateTime	Sí	No	No
updateTimestamp	Sí	No	No
wasNull	Sí	Sí	Sí

Tabla 53. Soporte para métodos ResultSet (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS

Notas:

1. No se da soporte agetBinaryStream en columnas de tipo CLOB.
2. 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.
3. El formato siguiente del método getObject no se soporta:

   ```
   getObject(int Índiceparámetros, java.util.Map
   map)
   ```

4. 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)
   ```

5. 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)
   ```

6. 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)
   ```

7. 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)
   ```

8. 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)
   ```

9. Esto es un método de JDBC 4.0.

Tabla 54. Soporte para métodos ResultSetMetaData

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
getCatalogName	Sí	Sí	Sí
getColumnClassName	No	Sí	Sí
getColumnCount	Sí	Sí	Sí
getColumnDisplaySize	Sí	Sí	Sí
getColumnLabel	Sí	Sí	Sí
getColumnName	Sí	Sí	Sí
getColumnType	Sí	Sí	Sí

Tabla 54. Soporte para métodos ResultSetMetaData (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
getColumnTypeName	Sí	Sí	Sí
getPrecision	Sí	Sí	Sí
getScale	Sí	Sí	Sí
getSchemaName	Sí	Sí	Sí
getTableName	Sí[1]	Sí	Sí
isAutoIncrement	Sí	Sí	Sí
isCaseSensitive	Sí	Sí	Sí
isCurrency	Sí	Sí	Sí
isDefinitelyWritable	Sí	Sí	Sí
isNullable	Sí	Sí	Sí
isReadOnly	Sí	Sí	Sí
isSearchable	Sí	Sí	Sí
isSigned	Sí	Sí	Sí
isWritable	Sí	Sí	Sí

Notas:

1. Para las fuentes de datos IDS, getTableName no devuelve un valor.

Tabla 55. Soporte para métodos RowId[1]

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ[2]	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
equals	Sí	No	No
getBytes	Sí	No	No
hashCode	No	No	No
toString	Sí	No	No

Notas:

1. Estos métodos son métodos de JDBC 4.0.
2. Estos métodos se pueden utilizar para conexiones con fuentes de datos DB2 para z/OS, DB2 para i5/OS e IDS.

Tabla 56. Soporte para métodos SQLClientInfoException[1]

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
Métodos heredados de java.lang.Exception	Sí	No	No
Métodos heredados de java.lang.Throwable	Sí	No	No
Métodos heredados de java.lang.Object	Sí	No	No
getFailedProperties	Sí	No	No

Tabla 56. Soporte para métodos SQLClientInfoException[1] (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS

Nota:

1. Esto es una clase de JDBC 4.0.

Tabla 57. Soporte para métodos SQLData

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
getSQLTypeName	No	No	No
readSQL	No	No	No
writeSQL	No	No	No

Tabla 58. Soporte para métodos SQLDataException[1]

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
Métodos heredados de java.lang.Exception	Sí	No	No
Métodos heredados de java.lang.Throwable	Sí	No	No
Métodos heredados de java.lang.Object	Sí	No	No

Nota:

1. Esto es una clase de JDBC 4.0.

Tabla 59. Soporte para métodos SQLException

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
Métodos heredados de java.lang.Exception	Sí	Sí	Sí
getSQLState	Sí	Sí	Sí
getErrorCode	Sí	Sí	Sí
getNextException	Sí	Sí	Sí
setNextException	Sí	Sí	Sí

Tabla 60. Soporte para métodos SQLFeatureNotSupported[1]

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
Métodos heredados de java.lang.Exception	Sí	No	No

Tabla 60. Soporte para métodos SQLFeatureNotSupported[1] (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
Métodos heredados de java.lang.Throwable	Sí	No	No
Métodos heredados de java.lang.Object	Sí	No	No

Nota:

1. Esto es una clase de JDBC 4.0.

Tabla 61. Soporte para métodos SQLInput

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
readArray	No	No	No
readAsciiStream	No	No	No
readBigDecimal	No	No	No
readBinaryStream	No	No	No
readBlob	No	No	No
readBoolean	No	No	No
readByte	No	No	No
readBytes	No	No	No
readCharacterStream	No	No	No
readClob	No	No	No
readDate	No	No	No
readDouble	No	No	No
readFloat	No	No	No
readInt	No	No	No
readLong	No	No	No
readObject	No	No	No
readRef	No	No	No
readShort	No	No	No
readString	No	No	No
readTime	No	No	No
readTimestamp	No	No	No
wasNull	No	No	No

Tabla 62. Soporte para métodos SQLIntegrityConstraintViolationException[1]

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
Métodos heredados de java.lang.Exception	Sí	No	No

Tabla 62. Soporte para métodos SQLIntegrityConstraintViolationException[1] (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
Métodos heredados de java.lang.Throwable	Sí	No	No
Métodos heredados de java.lang.Object	Sí	No	No

Nota:

1. Esto es una clase de JDBC 4.0.

Tabla 63. Soporte para métodos SQLInvalidAuthorizationSpecException[1]

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
Métodos heredados de java.lang.Exception	Sí	No	No
Métodos heredados de java.lang.Throwable	Sí	No	No
Métodos heredados de java.lang.Object	Sí	No	No

Nota:

1. Esto es una clase de JDBC 4.0.

Tabla 64. Soporte para métodos SQLNonTransientConnectionException[1]

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
Métodos heredados de java.lang.Exception	Sí	No	No
Métodos heredados de java.lang.Throwable	Sí	No	No
Métodos heredados de java.lang.Object	Sí	No	No

Nota:

1. Esto es una clase de JDBC 4.0.

Tabla 65. Soporte para métodos SQLNonTransientException[1]

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
Métodos heredados de java.lang.Exception	Sí	No	No
Métodos heredados de java.lang.Throwable	Sí	No	No
Métodos heredados de java.lang.Object	Sí	No	No

Nota:

1. Esto es una clase de JDBC 4.0.

Tabla 66. Soporte para métodos SQLOutput

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
writeArray	No	No	No
writeAsciiStream	No	No	No
writeBigDecimal	No	No	No
writeBinaryStream	No	No	No
writeBlob	No	No	No
writeBoolean	No	No	No
writeByte	No	No	No
writeBytes	No	No	No
writeCharacterStream	No	No	No
writeClob	No	No	No
writeDate	No	No	No
writeDouble	No	No	No
writeFloat	No	No	No
writeInt	No	No	No
writeLong	No	No	No
writeObject	No	No	No
writeRef	No	No	No
writeShort	No	No	No
writeString	No	No	No
writeStruct	No	No	No
writeTime	No	No	No
writeTimestamp	No	No	No

Tabla 67. Soporte para métodos SQLRecoverableException[1]

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
Métodos heredados de java.lang.Exception	Sí	No	No
Métodos heredados de java.lang.Throwable	Sí	No	No
Métodos heredados de java.lang.Object	Sí	No	No

Nota:

1. Esto es una clase de JDBC 4.0.

Tabla 68. Soporte para métodos SQLSyntaxErrorException[1]

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
Métodos heredados de java.lang.Exception	Sí	No	No
Métodos heredados de java.lang.Throwable	Sí	No	No
Métodos heredados de java.lang.Object	Sí	No	No

Nota:

1. Esto es una clase de JDBC 4.0.

Tabla 69. Soporte para métodos SQLTimeoutException[1]

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
Métodos heredados de java.lang.Exception	Sí	No	No
Métodos heredados de java.lang.Throwable	Sí	No	No
Métodos heredados de java.lang.Object	Sí	No	No

Nota:

1. Esto es una clase de JDBC 4.0.

Tabla 70. Soporte para métodos SQLTransientConnectionException[1]

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
Métodos heredados de java.lang.Exception	Sí	No	No
Métodos heredados de java.lang.Throwable	Sí	No	No
Métodos heredados de java.lang.Object	Sí	No	No

Nota:

1. Esto es una clase de JDBC 4.0.

Tabla 71. Soporte para métodos SQLTransientException[1]

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
Métodos heredados de java.lang.Exception	Sí	No	No
Métodos heredados de java.lang.Throwable	Sí	No	No
Métodos heredados de java.lang.Object	Sí	No	No

Tabla 71. Soporte para métodos SQLTransientException[1] (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS

Nota:

1. Esto es una clase de JDBC 4.0.

Tabla 72. Soporte para métodos SQLTransientRollbackException[1]

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
Métodos heredados de java.lang.Exception	Sí	No	No
Métodos heredados de java.lang.Throwable	Sí	No	No
Métodos heredados de java.lang.Object	Sí	No	No

Nota:

1. Esto es una clase de JDBC 4.0.

Tabla 73. Soporte para métodos SQLXML[1]

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
free	Sí	No	No
getBinaryStream	Sí	No	No
getCharacterStream	Sí	No	No
getSource	Sí	No	No
getString	Sí	No	No
setBinaryStream	Sí	No	No
setCharacterStream	Sí	No	No
setResult	Sí	No	No
setString	Sí	No	No

Notas:

1. Estos métodos son métodos de JDBC 4.0.

Tabla 74. Soporte para métodos Statement

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
addBatch	Sí	Sí	Sí
cancel	Sí[1]	Sí[2]	Sí
clearBatch	Sí	Sí	Sí
clearWarnings	Sí	Sí	Sí

Tabla 74. Soporte para métodos Statement (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
close	Sí	Sí	Sí
execute	Sí	Sí[3]	Sí[3]
executeBatch	Sí	Sí	Sí
executeQuery	Sí	Sí	Sí
executeUpdate	Sí	Sí[3]	Sí[3]
getConnection	Sí	Sí	Sí
getFetchDirection	Sí	Sí	Sí
getFetchSize	Sí	Sí	Sí
getGeneratedKeys	Sí	No	No
getMaxFieldSize	Sí	Sí	Sí
getMaxRows	Sí	Sí	Sí
getMoreResults	Sí	Sí[3]	Sí[3]
getQueryTimeout	Sí[2]	Sí	Sí
getResultSet	Sí	Sí	Sí
getResultSetConcurrency	Sí	Sí	Sí
getResultSetHoldability	Sí	No	No
getResultSetType	Sí	Sí	Sí
getUpdateCount[4]	Sí	Sí	Sí
getWarnings	Sí	Sí	Sí
isClosed[6]	Sí	No	No
isPoolable[6]	Sí	No	No
setCursorName	Sí	Sí	Sí
setEscapeProcessing	Sí	Sí	Sí
setFetchDirection	Sí	Sí	Sí
setFetchSize	Sí	Sí	Sí
setMaxFieldSize	Sí	Sí	Sí
setMaxRows	Sí	Sí	Sí
setPoolable[6]	Sí	No	No
setQueryTimeout	Sí[5]	Sí	Sí

Tabla 74. Soporte para métodos Statement (continuación)

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS

Notas:

1. 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 9 o posterior.
2. Con el controlador DB2 JDBC tipo 2 para Linux, UNIX y Windows, únicamente se da soporte a Statement.cancel() en los siguientes entornos:
   - Conexiones a un servidor DB2 Database para Linux, UNIX y Windows versión 8 o posterior.
   - Conexiones a un servidor DB2 para z/OS versión 9 o posterior.
3. 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.
4. No está soportado para conjuntos de resultados de procedimiento almacenado.
5. Para IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 2 sobre DB2 para z/OS y para DB2 para i5/OS, este método se puede utilizar solamente para un valor de *segundos* igual a 0.
6. Esto es un método de JDBC 4.0.

Tabla 75. Soporte para métodos Struct

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
getSQLTypeName	No	No	No
getAttributes	No	No	No

Tabla 76. Soporte para métodos Wrapper

Método JDBC[1]	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
isWrapperFor	Sí	No	No
unWrap	Sí	No	No

Notas:

1. Estos métodos son métodos de JDBC 4.0.

Tabla 77. Soporte para métodos javax.sql.XAConnection

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ[1]	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
Métodos heredados de javax.sql.PooledConnection	Sí	Sí	Sí
getXAResource	Sí	Sí	Sí

Notas:

1. Estos métodos se pueden utilizar para IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 2 con un servidor DB2 Database para Linux, UNIX y Windows o IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 4.

Tabla 78. Soporte para métodos XADataSource

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
getLoginTimeout	Sí	Sí	Sí
getLogWriter	Sí	Sí	Sí
getXAConnection	Sí	Sí	Sí
setLoginTimeout	Sí	Sí	Sí
setLogWriter	Sí	Sí	Sí

Tabla 79. Soporte para métodos javax.transaction.xa.XAResource

Método JDBC	Soporte de IBM Data Server Driver para JDBC y SQLJ	Soporte del controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows	Soporte del controlador JDBC de IDS
commit	Sí[1]	Sí	Sí
end	Sí[1]	Sí	Sí
forget	Sí[1]	Sí	Sí
getTransactionTimeout	Sí[2]	Sí	Sí
isSameRM	Sí[1]	Sí	Sí
prepare	Sí[1]	Sí	Sí
recover	Sí[1]	Sí	Sí
rollback	Sí[1]	Sí	Sí
setTransactionTimeout	Sí[2]	Sí	Sí
start	Sí[1]	Sí	Sí

Notas:

1. Estos métodos se pueden utilizar para IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 2 con un servidor DB2 Database para Linux, UNIX y Windows o IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 4.
2. Este método se puede utilizar para IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 4 con DB2 Database para Linux, UNIX y Windows Versión 9.1 o posterior.

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.

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.

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.

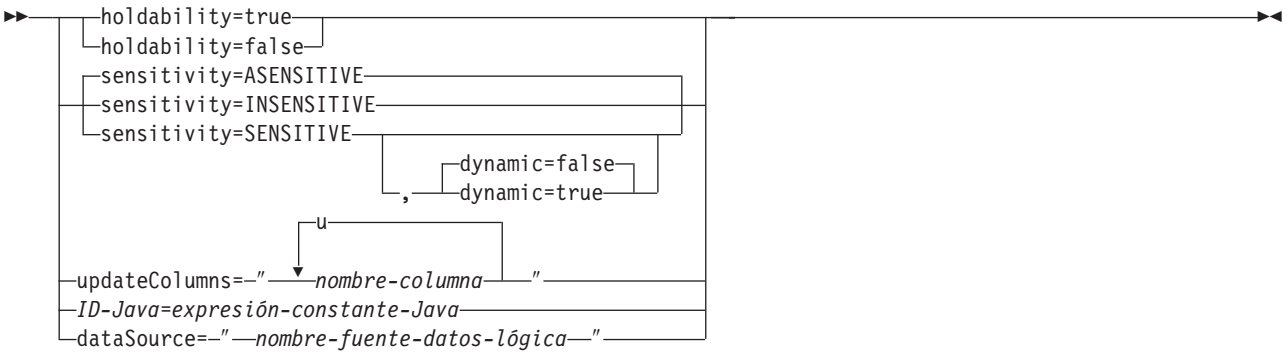
Es necesario que implemente sqlj.runtime.Scrollable cuando declare un iterador desplazable. Consulte ″Utilizar iteradores desplazables en una aplicación SQLJ″ para obtener información sobre iteradores desplazables.

Cláusula with de SQLJ

La cláusula with especifica uno o más atributos para un iterador o contexto de conexión.

Sintaxis

elemento-with:



Descripción

holdability
Para un iterador, especifica si el iterador conserva su posición en una tabla después de ejecutarse una operación COMMIT. El valor de holdability debe ser `true` o `false`.

sensitivity
Para un iterador, especifica si los cambios hechos en la tabla subyacente pueden ser visibles para el iterador después de abrir el iterador. El valor debe ser INSENSITIVE, SENSITIVE o ASENSITIVE. El valor por omisión es ASENSITIVE.

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.

updateColumns
Para un iterador, especifica las columnas que se deben modificar cuando el iterador se utiliza para una sentencia UPDATE de posición. El valor de updateColumns debe ser una cadena de caracteres literal que contenga los nombres de las columnas, separados por comas.

nombre-columna
Para un iterador, especifica una columna de la tabla de resultados que se debe actualizar utilizando el iterador.

ID-Java
Para un iterador o contexto de conexión, especifica una variable Java que identifica un atributo definido por el usuario del iterador o contexto de conexión. El valor de *expresión-constante-Java* también está definido por el usuario.

dataSource
Para un contexto de conexión, especifica el nombre lógico de un objeto

DataSource, creado por separado, que representa la fuente de datos a la que se conectará la aplicación. Esta opción solo está disponible para el IBM Data Server Driver para JDBC y SQLJ.

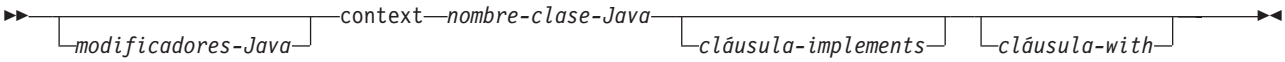
Notas de uso

- 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.

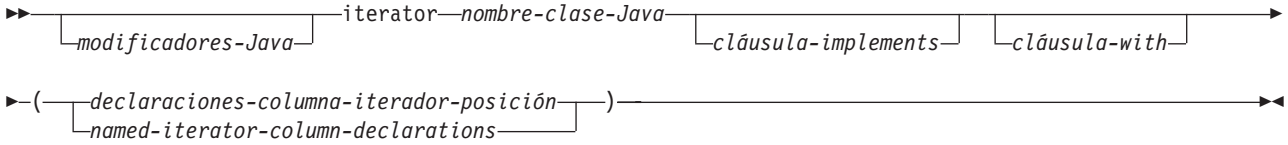
Cláusula de declaración de iterador de SQLJ

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

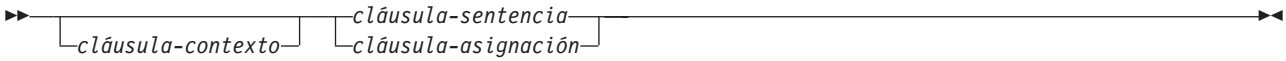
- Una cláusula de declaración de iterador puede aparecer en cualquier punto de un programa Java en donde pueda existir una declaración de clase Java.
- Cuando una declaración de iterador de nombre contiene más de un par de tipos de datos Java e ID de Java, todos los ID de Java de la lista tienen que 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

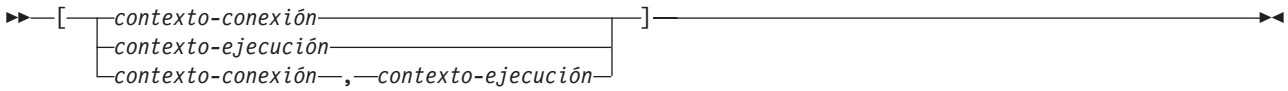
- Una cláusula ejecutable puede aparecer en cualquier lugar de un programa Java en que pueda aparecer una sentencia Java.
- SQLJ utiliza la clase java.sql.SQLException para notificar los códigos negativos de SQL resultantes de cláusulas ejecutables.

  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 las sentencias de SQL de DB2 Database para Linux, UNIX y Windows de Tabla 80 en la cláusula de una 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. Consulte "Cláusula SET TRANSACTION de SQLJ" para obtener más información.

Tabla 80. Sentencias de SQL válidas en una cláusula de sentencia de SQLJ
ALTER DATABASE
ALTER FUNCTION
ALTER INDEX
ALTER PROCEDURE
ALTER STOGROUP
ALTER TABLE

Tabla 80. Sentencias de SQL válidas en una cláusula de sentencia de SQLJ (continuación)

ALTER TABLESPACE
CALL
COMMENT ON
COMMIT
CREATE ALIAS
CREATE DATABASE
CREATE DISTINCT TYPE
CREATE FUNCTION
CREATE GLOBAL TEMPORARY TABLE
CREATE INDEX
CREATE PROCEDURE
CREATE STOGROUP
CREATE SYNONYM
CREATE TABLE
CREATE TABLESPACE
CREATE TRIGGER
CREATE VIEW
DECLARE GLOBAL TEMPORARY TABLE
DELETE
DROP ALIAS
DROP DATABASE
DROP DISTINCT TYPE
DROP FUNCTION
DROP INDEX
DROP PACKAGE
DROP PROCEDURE
DROP STOGROUP
DROP SYNONYM
DROP TABLE
DROP TABLESPACE
DROP TRIGGER
DROP VIEW
FETCH
GRANT
INSERT
LOCK TABLE
MERGE
REVOKE
ROLLBACK
SAVEPOINT
SELECT INTO
SET CURRENT DEFAULT TRANSFORM GROUP
SET CURRENT DEGREE
SET CURRENT EXPLAIN MODE
SET CURRENT EXPLAIN SNAPSHOT
SET CURRENT ISOLATION
SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION
SET CURRENT OPTIMIZATION HINT
SET CURRENT PACKAGESET (no se soporta USER)
SET CURRENT PRECISION
SET CURRENT QUERY OPTIMIZATION
SET CURRENT REFRESH AGE
SET CURRENT SCHEMA
SET PATH
UPDATE

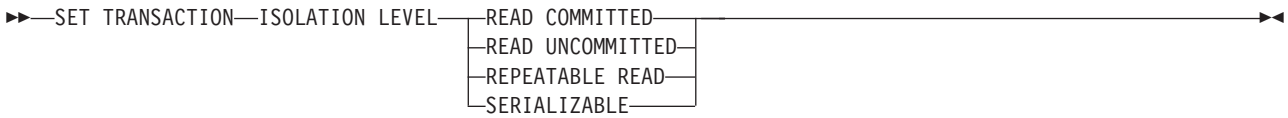
Notas de uso

- SQLJ da soporte a operaciones DELETE y UPDATE de posición y de búsqueda.
- Para una sentencia FETCH, una sentencia DELETE de posición o una sentencia UPDATE de posición, debe utilizar un iterador para apuntar a las filas de una tabla de resultados.

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.

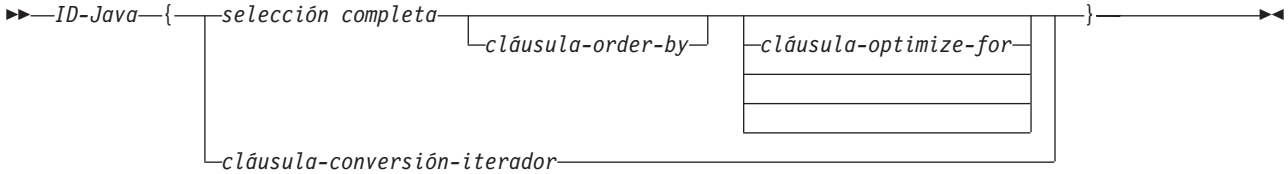
Notas de uso

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 conjunto de resultados de JDBC en un iterador.

Sintaxis

Descripción

expresión-lenguaje-principal
Identifica el conjunto de resultados de JDBC que se debe convertir en un iterador de SQLJ.

Notas de uso

- Si el iterador al que se debe convertir el conjunto de resultados es un iterador de posición, el número de columnas del conjunto de resultados debe coincidir con el número de columnas del iterador. Además, el tipo de datos de cada columna del conjunto de resultados debe ser compatible con el tipo de datos de la columna correspondiente del iterador.
- Si el iterador es un iterador de nombre, el nombre de cada método accesor debe coincidir con el nombre de una columna del conjunto de resultados, salvo en lo que respecta al uso de letras mayúsculas y minúsculas. Además, el tipo de datos

del objeto devuelto por un método accesor debe ser compatible con el tipo de datos de la columna correspondiente del conjunto de resultados.

- Cuando se cierra un iterador que se ha generado mediante una cláusula de conversión a iterador, también se cierra el conjunto de resultados a partir del cual se creó el iterador.

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 81. 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 conjunto de resultados 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 82. 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.
DefaultRuntime	Se implementa mediante SQLJ para satisfacer el comportamiento del tiempo de ejecución previsto de SQLJ para la mayoría de 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.
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.

Tabla 82. Resumen de las clases de sqlj.runtime (continuación)

Nombre de clase	Finalidad
SQLNullException	Deriva de la clase java.sql.SQLException. Se emite una excepción sqlj.runtime.SQLNullException cuando se inserta 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.

usuario
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 del 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:

usuario
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 interadores 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.

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.

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.

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.

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 cadena 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 excepción de SQL.

Suponga que *m* es el número de filas de la tabla de resultados y *x* es el número de fila actual de la tabla de resultados. Si *n*>0 y *x*+*n*>*m*, el iterador se sitúa 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.

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 los rows 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.

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.

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 para JDBC de IBM Data Server Driver para JDBC y SQLJ

El 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 83 resume las interfaces específicas de IBM Data Server Driver para JDBC y SQLJ.

Tabla 83. Resumen de las interfaces específicas de IBM Data Server Driver para JDBC y SQLJ proporcionadas por el IBM Data Server Driver para JDBC y SQLJ

Nombre de interfaz	Fuentes de datos aplicables	Finalidad
DB2Connection	1 en la página 331, 2 en la página 331, 3 en la página 331	Amplía la interfaz java.sql.Connection.
DB2DatabaseMetaData	1 en la página 331, 2 en la página 331, 3 en la página 331	Amplía la interfaz java.sql.DatabaseMetaData.
DB2Diagnosable	1 en la página 331, 2 en la página 331, 3 en la página 331	Proporciona un mecanismo para obtener los diagnósticos de DB2 a partir de un SQLException de DB2.
DB2PreparedStatement	1 en la página 331, 2 en la página 331, 3 en la página 331	Amplía las interfaces com.ibm.db2.jcc.DB2Statement y java.sql.PreparedStatement.
DB2RowID	1 en la página 331, 2 en la página 331	Se utiliza para declarar objetos Java para su utilización con el tipo de datos ROWID.
DB2Statement	1 en la página 331, 2 en la página 331, 3 en la página 331	Amplía la interfaz java.sql.Statement.

Tabla 83. Resumen de las interfaces específicas de IBM Data Server Driver para JDBC y SQLJ proporcionadas por el IBM Data Server Driver para JDBC y SQLJ (continuación)

Nombre de interfaz	Fuentes de datos aplicables	Finalidad
DB2TraceManagerMXBean	1, 2, 3	Proporciona la interfaz MBean para el controlador de rastreo remoto.
DB2SystemMonitor	1, 2, 3	Se utiliza para recoger datos de supervisión del sistema para una conexión.
DB2Xml	1, 2	Se utiliza para actualizar y recuperar datos de columnas XML.

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 Dynamic Server

La Tabla 84 resume las clases específicas de IBM Data Server Driver para JDBC y SQLJ.

Tabla 84. Resumen de las clases específicas de IBM Data Server Driver para JDBC y SQLJ proporcionadas por el 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 332	Las instancias de la clase DB2Administrator se utilizan para recuperar objetos DB2CataloguedDatabase.
DB2BaseDataSource	1 en la página 332, 2 en la página 332, 3 en la página 332	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.
DB2CataloguedDatabase	2 en la página 332	Contiene métodos para recuperar información sobre una base de datos DB2 Database para Linux, UNIX y Windows.
DB2ClientRerouteServerList	1 en la página 332, 2 en la página 332	Implementa las interfaces java.io.Serializable y javax.naming.Referenceable.
DB2ConnectionPoolDataSource	1 en la página 332, 2 en la página 332, 3 en la página 332	Fábrica de objetos PooledConnection.
DB2ExceptionFormatter	1 en la página 332, 2 en la página 332, 3 en la página 332	Contiene métodos para la impresión de la información de diagnóstico en una corriente.
DB2JCCPlugin	2 en la página 332	Clase abstracta para la implementación de plugins de seguridad JDBC.
DB2PooledConnection	1 en la página 332, 2 en la página 332, 3 en la página 332	Proporciona métodos que puede utilizar un servidor de aplicaciones para cambiar de usuarios en una conexión fiable preexistente.
DB2PoolMonitor	1 en la página 332, 2 en la página 332	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 en la página 332, 2 en la página 332, 3 en la página 332	Amplía la clase DataBaseDataSource. No soporta agrupaciones de conexiones ni transacciones distribuidas.

Tabla 84. Resumen de las clases específicas de IBM Data Server Driver para JDBC y SQLJ proporcionadas por el IBM Data Server Driver para JDBC y SQLJ (continuación)

Nombre de clase	Fuentes de datos aplicables	Finalidad
DB2Sqlca	1, 2, 3	Encapsulación de la SQLCA de DB2.
DB2TraceManager	1, 2, 3	Controla el grabador de anotaciones cronológicas globales.
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).

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 Dynamic Server

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.

Clase 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 el IBM Data Server Driver para JDBC y SQLJ.

Cada una de estas propiedades tiene un método setXXX para definir el valor de la propiedad y un método getXXX para obtener el valor. Un método setXXX tiene este formato:

```
void setNombre-propiedad(tipo-datos valor-propiedad)
```

Un método getXXX tiene este formato:

```
tipo-datos getNombre-propiedad()
```

Nombre-propiedad es el nombre de propiedad no calificado. En el caso de propiedades que no son específicas de IBM Informix Dynamic Server (IDS), el primer carácter del nombre de la propiedad está en mayúsculas. En el caso de propiedades que únicamente utiliza IDS, todos los caracteres del nombre de la propiedad están en mayúsculas.

La tabla siguiente lista las propiedades del IBM Data Server Driver para JDBC y SQLJ y los tipos de datos asociados a ellas.

Tabla 85. 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 338	String
com.ibm.db2.jcc.DB2BaseDataSource.blockingReadConnectionTimeout	1 en la página 338, 2 en la página 338, 3 en la página 338	int
com.ibm.db2.jcc.DB2BaseDataSource.charOutputSize	1 en la página 338	short
com.ibm.db2.jcc.DB2BaseDataSource.clientAccountingInformation (IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 2 sobre DB2 para z/OS solamente)	1 en la página 338, 2 en la página 338	String
com.ibm.db2.jcc.DB2BaseDataSource.clientApplicationInformation (IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 2 sobre DB2 para z/OS solamente)	1 en la página 338	String
com.ibm.db2.jcc.DB2BaseDataSource.clientDebugInfo (IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 2 sobre DB2 para z/OS solamente)	1 en la página 338	String
com.ibm.db2.jcc.DB2BaseDataSource.clientProgramId (IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 2 sobre DB2 para z/OS solamente)	1 en la página 338	String
com.ibm.db2.jcc.DB2BaseDataSource.clientProgramName	1 en la página 338	String
com.ibm.db2.jcc.DB2BaseDataSource.clientRerouteAlternateServerName	1 en la página 338, 2 en la página 338	String
com.ibm.db2.jcc.DB2BaseDataSource.clientRerouteAlternatePortNumber	1 en la página 338, 2 en la página 338	String
com.ibm.db2.jcc.DB2BaseDataSource.clientRerouteServerListJNDIContext	1 en la página 338, 2 en la página 338	javax.naming.Context
com.ibm.db2.jcc.DB2BaseDataSource.clientRerouteServerListJNDIName	1 en la página 338, 2 en la página 338	String
com.ibm.db2.jcc.DB2BaseDataSource.clientUser (IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 2 sobre DB2 para z/OS solamente)	1 en la página 338	String
com.ibm.db2.jcc.DB2BaseDataSource.clientWorkstation (IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 2 sobre DB2 para z/OS solamente)	1 en la página 338	String

Tabla 85. 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.connectNode	2 en la página 338	int
com.ibm.db2.jcc.DB2BaseDataSource.currentDegree	1 en la página 338, 2 en la página 338	String
com.ibm.db2.jcc.DB2BaseDataSource.currentExplainMode	2 en la página 338	String
com.ibm.db2.jcc.DB2BaseDataSource.currentExplainSnapshot	2 en la página 338	String
com.ibm.db2.jcc.DB2BaseDataSource.currentFunctionPath	1 en la página 338, 2 en la página 338	String
com.ibm.db2.jcc.DB2BaseDataSource.currentLockTimeout	2 en la página 338, 3 en la página 338	int
com.ibm.db2.jcc.DB2BaseDataSource.currentMaintainedTableTypesForOptimization	1 en la página 338, 2 en la página 338	String
com.ibm.db2.jcc.DB2BaseDataSource.currentPackagePath	1 en la página 338, 2 en la página 338	String
com.ibm.db2.jcc.DB2BaseDataSource.currentPackageSet	1 en la página 338, 2 en la página 338	String
com.ibm.db2.jcc.DB2BaseDataSource.currentQueryOptimization	2 en la página 338	int
com.ibm.db2.jcc.DB2BaseDataSource.currentRefreshAge	1 en la página 338, 2 en la página 338	long
com.ibm.db2.jcc.DB2BaseDataSource.currentSchema	1 en la página 338, 2 en la página 338	String
com.ibm.db2.jcc.DB2BaseDataSource.cursorSensitivity	1 en la página 338, 2 en la página 338	int
com.ibm.db2.jcc.DB2BaseDataSource.currentSQLID	1 en la página 338	String
com.ibm.db2.jcc.DB2BaseDataSource.databaseName	1 en la página 338, 2 en la página 338, 3 en la página 338	String
com.ibm.db2.jcc.DB2BaseDataSource.dateFormat	1 en la página 338, 2 en la página 338	int
com.ibm.db2.jcc.DB2BaseDataSource.decimalRoundingMode	1 en la página 338, 2 en la página 338	int
com.ibm.db2.jcc.DB2BaseDataSource.defaultIsolationLevel	1 en la página 338, 2 en la página 338, 3 en la página 338	int
com.ibm.db2.jcc.DB2BaseDataSource.deferPrepares	1 en la página 338, 2 en la página 338, 3 en la página 338	boolean

Tabla 85. 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.description	1 en la página 338, 2 en la página 338, 3 en la página 338	String
com.ibm.db2.jcc.DB2BaseDataSource.downgradeHoldCursorsUnderXa	1 en la página 338, 2 en la página 338,3 en la página 338	boolean
com.ibm.db2.jcc.DB2BaseDataSource.driverType	1 en la página 338, 2 en la página 338, 3 en la página 338	int
com.ibm.db2.jcc.DB2BaseDataSource.enableConnectionConcentrator	1 en la página 338	boolean
com.ibm.db2.jcc.DB2BaseDataSource.enableSysplexWLB	1 en la página 338	boolean
com.ibm.db2.jcc.DB2BaseDataSource.fullyMaterializeInputStreams	1 en la página 338, 2 en la página 338	boolean
com.ibm.db2.jcc.DB2BaseDataSource.fullyMaterializeLobData	1 en la página 338, 2 en la página 338, 3 en la página 338	boolean
com.ibm.db2.jcc.DB2BaseDataSource.gssCredential	1 en la página 338, 2 en la página 338	Object
com.ibm.db2.jcc.DB2BaseDataSource.jdbcCollection	1 en la página 338	String
com.ibm.db2.jcc.DB2BaseDataSource.keepDynamic	1 en la página 338, 3 en la página 338	int
com.ibm.db2.jcc.DB2BaseDataSource.kerberosServerPrincipal	1 en la página 338, 2 en la página 338	String
com.ibm.db2.jcc.DB2BaseDataSource.loginTimeout (no válido para IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 2 sobre DB2 para z/OS)	1 en la página 338, 2 en la página 338, 3 en la página 338	int
com.ibm.db2.jcc.DB2BaseDataSource.logWriter	1 en la página 338, 2 en la página 338, 3 en la página 338	PrintWriter
com.ibm.db2.jcc.DB2BaseDataSource.maxRetriesForClientReroute	1 en la página 338, 2 en la página 338	int
com.ibm.db2.jcc.DB2BaseDataSource.maxTransportObjects	1 en la página 338	int
com.ibm.db2.jcc.DB2BaseDataSource.optimizationProfile	2 en la página 338	String
com.ibm.db2.jcc.DB2BaseDataSource.optimizationProfileToFlush	2 en la página 338	String
com.ibm.db2.jcc.DB2BaseDataSource.password	1 en la página 338, 2 en la página 338, 3 en la página 338	String

Tabla 85. 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.pkList (IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 2)	1 en la página 338	String
com.ibm.db2.jcc.DB2BaseDataSource.planName (solamente para IBM Data Server Driver para conectividad JDBC y SQLJ de tipo 2)	1 en la página 338	String
com.ibm.db2.jcc.DB2BaseDataSource.plugin	2 en la página 338	Object
com.ibm.db2.jcc.DB2BaseDataSource.pluginName	2 en la página 338	String
com.ibm.db2.jcc.DB2BaseDataSource.portNumber	1 en la página 338, 2 en la página 338, 3 en la página 338	int
com.ibm.db2.jcc.DB2BaseDataSource.progressiveStreaming	1 en la página 338, 2 en la página 338	int
com.ibm.db2.jcc.DB2BaseDataSource.queryDataSize	1 en la página 338, 2 en la página 338, 3 en la página 338	int
com.ibm.db2.jcc.DB2BaseDataSource.queryCloseImplicit	1 en la página 338, 3 en la página 338	int
com.ibm.db2.jcc.DB2BaseDataSource.readOnly	1 en la página 338, 2 en la página 338	boolean
com.ibm.db2.jcc.DB2BaseDataSource.resultSetHoldability	1 en la página 338, 2 en la página 338,3 en la página 338	int
com.ibm.db2.jcc.DB2BaseDataSource.resultSetHoldabilityForCatalogQueries	1 en la página 338, 2 en la página 338	int
com.ibm.db2.jcc.DB2BaseDataSource.retrieveMessagesFromServerOnGetMessage	1 en la página 338, 3 en la página 338	boolean
com.ibm.db2.jcc.DB2BaseDataSource.retryIntervalForClientReroute	1 en la página 338, 2 en la página 338	int
com.ibm.db2.jcc.DB2BaseDataSource.returnAlias	1 en la página 338, 2 en la página 338	short
com.ibm.db2.jcc.DB2BaseDataSource.securityMechanism	1 en la página 338, 2 en la página 338, 3 en la página 338	int
com.ibm.db2.jcc.DB2BaseDataSource.sendCharInputsUTF8	1 en la página 338	int
com.ibm.db2.jcc.DB2BaseDataSource.sendDataAsIs	1 en la página 338, 2 en la página 338, 3 en la página 338	boolean
com.ibm.db2.jcc.DB2BaseDataSource.serverName	1 en la página 338, 2 en la página 338, 3 en la página 338	String

Tabla 85. 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.sqljEnableClassLoaderSpecificProfiles	1 en la página 338	boolean
com.ibm.db2.jcc.DB2BaseDataSource.sslConnection	1 en la página 338, 2 en la página 338	boolean
com.ibm.db2.jcc.DB2BaseDataSource.streamBufferSize	1 en la página 338, 2 en la página 338	int
com.ibm.db2.jcc.DB2BaseDataSource.supportsAsynchronousXARollback	1 en la página 338, 2 en la página 338	int
com.ibm.db2.jcc.DB2BaseDataSource.sysSchema	1 en la página 338, 2 en la página 338	String
com.ibm.db2.jcc.DB2BaseDataSource.timeFormat	1 en la página 338, 2 en la página 338	int
com.ibm.db2.jcc.DB2BaseDataSource.traceDirectory	1 en la página 338, 2 en la página 338, 3 en la página 338	String
com.ibm.db2.jcc.DB2BaseDataSource.traceFile	1 en la página 338, 2 en la página 338, 3 en la página 338	String
com.ibm.db2.jcc.DB2BaseDataSource.traceFileAppend	1 en la página 338, 2 en la página 338, 3 en la página 338	boolean
com.ibm.db2.jcc.DB2BaseDataSource.traceLevel	1 en la página 338, 2 en la página 338, 3 en la página 338	int
com.ibm.db2.jcc.DB2BaseDataSource.useCachedCursor	1 en la página 338, 2 en la página 338	boolean
com.ibm.db2.jcc.DB2BaseDataSource.useJDBC4ColumnNameAndLabelSemantics	1 en la página 338, 2 en la página 338	int
com.ibm.db2.jcc.DB2BaseDataSource.user	1 en la página 338, 2 en la página 338, 3 en la página 338	String
com.ibm.db2.jcc.DB2BaseDataSource.useRowsetCursor	1 en la página 338	boolean
com.ibm.db2.jcc.DB2BaseDataSource.useTransactionRedirect	2 en la página 338	boolean
com.ibm.db2.jcc.DB2BaseDataSource.xaNetworkOptimization	1 en la página 338, 2 en la página 338, 3 en la página 338	boolean
com.ibm.db2.jcc.DB2BaseDataSource.DBANSIWARN	3 en la página 338	boolean
com.ibm.db2.jcc.DB2BaseDataSource.DBDATE	3 en la página 338	String

Tabla 85. 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	String
com.ibm.db2.jcc.DB2BaseDataSource.DBSPACETEMP	3	String
com.ibm.db2.jcc.DB2BaseDataSource.DBTEMP	3	String
com.ibm.db2.jcc.DB2BaseDataSource.DBUPSPACE	3	String
com.ibm.db2.jcc.DB2BaseDataSource.DELIMIDENT	3	boolean
com.ibm.db2.jcc.DB2BaseDataSource.IFX_DIRECTIVES	3	String
com.ibm.db2.jcc.DB2BaseDataSource.IFX_EXTDIRECTIVES	3	String
com.ibm.db2.jcc.DB2BaseDataSource.IFX_UPDDESC	3	String
com.ibm.db2.jcc.DB2BaseDataSource.IFX_XASTDCOMPLIANCE_XAEND	3	String
com.ibm.db2.jcc.DB2BaseDataSource.INFORMIXOPCACHE	3	String
com.ibm.db2.jcc.DB2BaseDataSource.INFORMIXSTACKSIZE	3	String
com.ibm.db2.jcc.DB2BaseDataSource.NODEFDAC	3	String
com.ibm.db2.jcc.DB2BaseDataSource.OPTCOMPIND	3	String
com.ibm.db2.jcc.DB2BaseDataSource.OPTOFC	3	String
com.ibm.db2.jcc.DB2BaseDataSource.PDQPRIORITY	3	String
com.ibm.db2.jcc.DB2BaseDataSource.PSORT_DBTEMP	3	String
com.ibm.db2.jcc.DB2BaseDataSource.PSORT_NPROCS	3	String
com.ibm.db2.jcc.DB2BaseDataSource.STMT_CACHE	3	String

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 Dynamic Server

Métodos de DB2BaseDataSource

Además de los métodos getXXX y setXXX para las propiedades de DB2BaseDataSource, están definidos los métodos siguientes solo para el 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 JNDI, que se encuentra en:

`http://java.sun.com/products/jndi/docs.html`

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)
```

Recupera 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 sólo se definen para el 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. nombreTipo puede ser un tipo de datos incorporado o un tipo diferenciado.

elements
Elementos utilizados para llenar el objeto Array.

deregisterDB2XmlObject
Formatos:

```
public void deregisterDB2XmlObject(String sqlIdSchema,
  String sqlIdName)
  throws SQLException
```

Elimina un esquema XML registrado previamente de la fuente de datos. deregisterDB2XmlObject invoca el procedimiento almacenado SYSPROC.XSR_REMOVE para eliminar el esquema XML.

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 un nombre de esquema SQL cualquiera. 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 tal caso, el sistema de bases de datos generará el valor para sqlIdName.

Este método no es aplicable a conexiones con fuentes de datos IBM Informix Dynamic Server.

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.

Este método no es aplicable a conexiones con fuentes de datos IBM Informix Dynamic Server.

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.

Este método no es aplicable a conexiones con fuentes de datos IBM Informix Dynamic Server.

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.

Este método no es aplicable a conexiones con fuentes de datos IBM Informix Dynamic Server.

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.

Este método no es aplicable a conexiones con fuentes de datos IBM Informix Dynamic Server.

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.

Este método no es aplicable a conexiones con fuentes de datos IBM Informix Dynamic Server.

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.

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 Seguridad Kerberos

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 del IBM Data Server Driver para JDBC y SQLJ puede tener un supervisor del sistema individual.

getJccLogWriter

Formato:

```
public PrintWriter getJccLogWriter()
  throws java.sql.SQLException
```

Obtiene el destino de rastreo actual para la función de rastreo del IBM Data Server Driver para JDBC y SQLJ.

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 Dynamic Server.

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.isValid.

isDB2GatewayConnection

Formato:

```
public boolean DB2Connection.isDB2GatewayConnection()
  throws java.sql.SQLException
```

Devuelve el valor `true` si la conexión con la fuente de datos se realiza a través de una pasarela DB2 Connect intermedia. En caso contrario devuelve `false`.

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 86.

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.

Este método no es aplicable a conexiones con fuentes de datos IBM Informix Dynamic Server.

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.

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
```

Proporciona uno o más documentos de esquema XML para registrar un esquema XML en la base de datos.`registerDB2XmlSchema` invoca los procedimientos almacenados SYSPROC.XSR_REGISTER, SYSPROC.XSR_ADDSCHEMADOC y SYSPROC.XSR_COMPLETE para registrar un esquema XML en 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 un nombre de esquema SQL cualquiera. 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 tal caso, el sistema de bases de datos generará el valor para 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 1000 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 30MB. Los valores no pueden ser nulos.

xmlSchemaDocumentsLengths
Longitudes de los documentos de esquema XML del parámetro

xmlSchemaDocumentsLengths, 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 esquema 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 de tipo 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 esquema 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. Cada valor xmlSchemaPropertiesLengths es un valor de tipo int.

isUsedForShredding
Indica si existen anotaciones en el esquema que debe utilizarse para la descomposición XML. isUsedForShredding es un valor boolean.

Este método no es aplicable a conexiones con fuentes de datos IBM Informix Dynamic Server.

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. UpdateDB2XmlSchema invoca el procedimiento almacenado SYSPROC.XSR_XSR_UPDATE para actualizar el esquema XML. 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 esquema 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 tipo String cuya longitud máxima es 128 bytes.

targetSqlIdName
Nombre del esquema XML registrado que se debe actualizar. targetSqlIdName es un valor de tipo String cuya longitud máxima es 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 tipo String cuya longitud máxima es 128 bytes.

sourceSqlIdName
Nombre del esquema XML registrado que se utiliza para actualizar el esquema XML de destino. sourceSqlIdName es un valor de tipo String cuya longitud máxima es 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 de tipo boolean. `false` es el valor por omisión.

Este método no es aplicable a conexiones con fuentes de datos IBM Informix Dynamic Server.

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 Dynamic Server.

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 *schema*, 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 Dynamic Server.

reuseDB2Connection (trusted connection reuse)

Formatos:

```
public Connection reuseDB2Connection(byte[] cookie,
  String user,
  String password,
  String usernameRegistry,
  byte[] userSecToken,
  String originalUser,
  java.util.Properties properties)
  throws java.sql.SQLException
public Connection reuseDB2Connection(byte[] cookie,
  org.ietf.GSSCredential gssCredential,
  String usernameRegistry,
  byte[] userSecToken,
  String originalUser,
  java.util.Properties properties)
  throws java.sql.SQLException
```

Método utilizado por un servidor de aplicaciones fiable 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.

La segunda forma de reuseDB2Connection sólo está soportada para la IBM Data Server Driver para conectividad JDBC y SQLJ 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 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 autenticado al usuario, el servidor de aplicaciones deberá pasar un ID de cliente que represente a un usuario sin autenticación.

password
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 DB2Connection reuseDB2Connection(String user,
  String password,
  java.util.Properties properties)
  throws java.sql.SQLException
public DB2Connection reuseDB2Connection(
  org.ietf.jgss.GSSCredential gssCredential,
  java.util.Properties properties)
  throws java.sql.SQLException
```

En un entorno de agrupación heterogéneo, se utiliza una instancia de Connection después de que se haya vuelto a realizar la autenticación.

Descripción de parámetros:

user
ID de autorización que se utiliza para establecer la conexión.

password
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 DB2Connection 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.

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 Java (`""`) es válida para este valor de parámetro, pero un valor `null` de Java no es válido.

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.

Este método no es aplicable a conexiones con fuentes de datos IBM Informix Dynamic Server.

setDB2ClientDebugInfo

Formatos:

```
public void setDB2ClientDebugInformation(String debugInfo)
  throws java.sql.SQLException
public void setDB2ClientDebugInformation(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.

El método setDB2ClientDebugInfo solamente es aplicable a conexiones con sistemas de bases de datos DB2 para z/OS.

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.

M*ip: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.

Este método no es aplicable a conexiones con fuentes de datos IBM Informix Dynamic Server.

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. 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.

Este método no es aplicable a conexiones con fuentes de datos IBM Informix Dynamic Server.

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 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 16 bytes. Una serie vacía Java (`""`) es válida para este valor de parámetro, pero un valor `null` de Java no es válido.

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.

Este método no es aplicable a conexiones con fuentes de datos IBM Informix Dynamic Server.

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 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 Java ("") es válida para este valor de parámetro, pero un valor `null` de Java no es válido.

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.

Este método no es aplicable a conexiones con fuentes de datos IBM Informix Dynamic Server.

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 del 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.

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 del 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 el 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 ″Propiedades del IBM Data Server Driver para JDBC y SQLJ″ para conocer los valores válidos.

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 del IBM Data Server Driver para JDBC y SQLJ y los tipos de datos asociados a ellas.

Tabla 87. 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 user,
  String password,
  java.util.Properties properties)
  throws java.sql.SQLException
public DB2PooledConnection getDB2PooledConnection(
  org.ietf.jgss.GSSCredential gssCredential,
  java.util.Properties propiedades)
  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.

password
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.

getDB2TrustedPooledConnection

Formatos:

```
public Object[] getDB2TrustedPooledConnection(String user,
  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. 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.

password
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.

properties
Propiedades de la conexión.

Interfaz DB2DatabaseMetaData

La interfaz com.ibm.db2.jcc.DB2DatabasemetaData amplía la interfaz java.sql.DatabaseMetaData.

DB2DatabaseMetaData implementa la interfaz java.sql.Wrapper.

Métodos DB2DatabaseMetaData:

Los métodos siguientes se definen solamente para el IBM Data Server Driver para JDBC y SQLJ.

isIDSDatabaseAnsiCompliant
Formato:

```
public boolean isIDSDatabaseAnsiCompliant();
```

Devuelve `true` si la base de datos activa actual de IBM Informix Dynamic Server (IDS) 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 IDS. Se emite una excepción de SQL (SQLException) si la fuente de datos no es una fuente de datos IDS.

isIDSDatabaseLogging
Formato:

```
public boolean isIDSDatabaseLogging();
```

Devuelve `true` si la base de datos IDS activa actual es compatible con el registro cronológico. En otro caso, devuelve false.

Una base de datos IDS compatible con el registro cronológico 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 IDS. Se emite una excepción de SQL (SQLException) si la fuente de datos no es una fuente de datos IDS.

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 sólo se definen para el 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 un 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 un 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 un 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 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 solamente para el 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
Destino de la información de diagnóstico.

header
Información definida por el usuario que se imprime al comienzo de la información de salida.

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 solamente para el 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.

password
Contraseña del *usuario*.

returnedToken

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.

Métodos DB2PooledConnection

Los métodos siguientes sólo se definen 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.

La segunda forma de reuseDB2Connection sólo está soportada para la IBM Data Server Driver para conectividad JDBC y SQLJ 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.

password
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.

properties
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.

password
Contraseña de *user*.

properties
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:

properties
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 el 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 sólo se definen 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 el 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 el 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 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 el 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 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.

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 el 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.

Métodos DB2PreparedStatement

Los métodos siguientes se definen solamente para el 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 Dynamic Server.

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.

Métodos DB2ResultSet:

Los métodos siguientes se definen solamente para el IBM Data Server Driver para JDBC y SQLJ.

getDB2RowChangeToken

Formato:

```
public long DB2ResultSet.getDB2RowChangeToken()
  throws java.sql.SQLException
```

Devuelve el distintivo 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.

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.

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`.

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 solamente para el 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 distintivo de cambio puede no tener la granularidad necesaria para evitar negativos falsos.

2 Existen columnas de bloqueo optimista disponibles, y el distintivo 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 se deriva de 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 ResultSet 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
Nombre de una columna del conjunto de resultados (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.

Métodos DB2RowID

El método siguiente sólo se define para el 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 del IBM Data Server Driver para JDBC y SQLJ.

DB2SimpleDataSource implementa la interfaz java.sql.Wrapper.

Propiedades de DB2SimpleDataSource

La propiedad siguiente se define solamente 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 esta propiedad.
String com.ibm.db2.jcc.DB2SimpleDataSource.password

Métodos DB2SimpleDataSource

El método siguiente sólo se define para el IBM Data Server Driver para JDBC y SQLJ.

setPassword
Formato:

```
public void setPassword(String password)
```

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 solamente para el 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 cadena de caracteres que contiene los valores SQLERRMC de SQLCA, delimitados por espacios.

getSqlErrmcTokens

Formato:

```
public abstract String[] getSqlErrmcTokens()
```

Devuelve una matriz, cuyos elementos contienen un símbolo SQLERRMC de SQLCA.

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 el 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 solamente para el 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 88.

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.

Tabla 88. (continuación)

Valor	Descripción
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.

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.

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. El identificador de programa es una serie de caracteres de 80 bytes que se utiliza para identificar al peticionario del programa. El servidor DB2 para z/OS coloca la serie de caracteres en registros de rastreo IFCID 316 junto con otras estadísticas, para que el usuario pueda identificar qué programa está asociado a una determinada sentencia de SQL.

getIDSSerial

Formato:

```
public int getIDSSerial()
  throws java.sql.SQLException
```

Recupera una clave que se ha creado automáticamente en la columna SERIAL de la sentencia INSERT ejecutada anteriormente.

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 es aplicable solamente a conexiones con bases de datos IBM Informix Dynamic Server (IDS).

getIDSSerial8

Formato:

```
public long getIDSSerial8()
  throws java.sql.SQLException
```

Recupera una clave que se ha creado automáticamente en la columna SERIAL8 de la sentencia INSERT ejecutada anteriormente.

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 IDS.

getIDSSQLStatementOffSet

Formato:

```
public int getIDSSQLStatementOffSet()
  throws java.sql.SQLException
```

Después de que una sentencia de SQL se ejecute en una fuente de datos IDS, 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 IDS.

Este método solamente es aplicable a conexiones con fuentes de datos IDS.

Interfaz DB2SystemMonitor

La interfaz com.ibm.db2.jcc.DB2SystemMonitor se utiliza para recoger datos de supervisión del sistema correspondientes a una conexión. Cada conexión puede tener una sola instancia de DB2SystemMonitor.

Campos de DB2SystemMonitor

Los campos siguientes se definen solamente para el 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 solamente para el 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 tiempo se inicializan cuando se invoca enable.

getApplicationTimeMillis

Formato:

```
public long getApplicationTimeMillis()
  throws java.sql.SQLException
```

Devuelve la suma de los tiempos transcurridos correspondientes a la aplicación, el controlador JDBC, la E/S de red y el servidor 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 de intervalo
Es el momento en el que se invoca start.

Fin de intervalo
Es el momento en el que se invoca stop.

getApplicationTimeMillis devuelve un 0 si la supervisión del sistema está inhabilitada. Si se invoca este método sin invocar primero stop, se produce una excepción SQLException.

getCoreDriverTimeMicros

Formato:

```
public long getCoreDriverTimeMicros()
  throws java.sql.SQLException
```

Devuelve la suma de los tiempos transcurridos de API supervisada que se recogieron mientras la supervisión del sistema estaba habilitada. El tiempo está expresado en microsegundos.

Una API supervisada es un método de controlador JDBC para la cual se obtiene el tiempo de proceso. En general, los tiempos transcurridos 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 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 de intervalo
Es el momento en el que la aplicación llama a la API supervisada.

Fin de intervalo
Inmediatamente antes de este momento la API supervisada devuelve el control a la aplicación.

getCoreDriverTimeMicros devuelve un 0 si la supervisión del sistema está inhabilitada. Si se invoca este método sin primero llamar al método stop, o si se invoca este método cuando la JVM subyacente no da soporte a la notificación de tiempos en microsegundos, se produce una excepción SQLException.

getNetworkIOTimeMicros

Formato:

```
public long getNetworkIOTimeMicros()
  throws java.sql.SQLException
```

Devuelve la suma de los tiempos transcurridos de E/S de red que se recogieron mientras la supervisión del sistema estaba habilitada. El tiempo está expresado en microsegundos.

El tiempo transcurrido de E/S de red incluye el tiempo que se invierte en escribir y leer datos de DRDA procedentes de corrientes de E/S de la red. Un intervalo de tiempo transcurrido de E/S de red es el intervalo de tiempo que se invierte en realizar las operaciones siguientes en el controlador JDBC:

- Emitir un mandato 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.

El tiempo transcurrido del servidor de bases de datos se define como el tiempo transcurrido necesario 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.

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 registro actuales y futuros de DriverManager, programas de registro de DataSource o programas de registro 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 registro.

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 conectividad JDBC y SQLJ 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 deben añadir o sobrescribir datos en el archivo especificado por medio del parámetro traceFile. `true` significa que no se sobregrabará el archivo existente.

unsetLogWriter

Formato:

```
public abstract void unsetLogWriter()
  throws java.sql.SQLException
```

Inhabilita la alteración temporal del programa de registro global 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. De lo contrario, getTraceFile devuelve un 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 traceFile, se escribe un solo archivo de registro.

traceFile puede incluir una vía de acceso de directorio.

fileAppend
Especifica si se añade o se sobregraba el archivo que especifica el 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 conectividad JDBC y SQLJ 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 conectividad JDBC y SQLJ 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 otro caso, getTraceDirectory devuelve un valor 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.

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. 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.

password
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.

properties
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. 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:

properties
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.

password
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.

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 el 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 un 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.

Diferencias en JDBC entre el controlador actual IBM Data Server Driver para JDBC y SQLJ y controladores JDBC de DB2 anteriores

Antes de migrar sus aplicaciones JDBC desde controladores más antiguos al IBM Data Server Driver para JDBC y SQLJ, es necesario que comprenda las diferencias entre esos controladores.

Importante: El controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows (controlador JDBC de DB2 de tipo 2) está en desuso. Esta información se ofrece para ayudarle a trasladar sus aplicaciones al IBM Data Server Driver para JDBC y SQLJ.

Métodos soportados

Para ver una comparación de los métodos que se pueden utilizar con los controladores JDBC, consulte ″Soporte de controlador para las API de JDBC″.

Utilización de la modalidad continua progresiva por los controladores JDBC

Para el IBM Data Server Driver para JDBC y SQLJ Versión 3.50 y versiones posteriores, la modalidad continua progresiva es el comportamiento por omisión para recuperar objetos LOB en las conexiones con DB2 Database para Linux, UNIX y Windows Versión 9.5 y versiones posteriores.

La modalidad continua progresiva se puede utilizar en el IBM Data Server Driver para JDBC y SQLJ Versión 3.1 y versiones posteriores, pero para el IBM Data Server Driver para JDBC y SQLJ Versión 3.2 y versiones posteriores, 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 versiones posteriores.

Las versiones anteriores del IBM Data Server Driver para JDBC y SQLJ y del controlador JDBC de DB2 de tipo 2 no son compatibles con 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

Los valores de ResultSetMetaData.getColumnName y ResultSetMetaData.getColumnLabel para IBM Data Server Driver para JDBC y SQLJ Versión 4.0 y posterior difieren de los valores devueltos para versiones anteriores del IBM Data Server Driver para JDBC y SQLJ o para otros controladores JDBC de IBM.

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 de proceso por lotes con claves generadas automáticamente causan una excepción de SQL.

Cuando se utiliza IBM Data Server Driver para JDBC y SQLJ Versión 3.50 o posterior, el preparar una sentencia de SQL para recuperar claves generadas automáticamente y al utilizar el objeto PreparedStatement para actualizaciones de proceso por lotes produce un 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.

Soporte para conjuntos de resultados desplazables y actualizables

El IBM Data Server Driver para JDBC y SQLJ soporta conjuntos de resultados desplazables y actualizables.

El controlador JDBC de DB2 de tipo 2 es compatible con ResultSet desplazables, pero no con ResultSet actualizables.

Diferencia en la sintaxis del URL

La sintaxis del parámetro *url* en el método DriverManager.getConnection es diferente para cada controlador. Consulte los temas siguientes para obtener más información:

- ″Conectar con una fuente de datos utilizando la interfaz DriverManager con IBM Data Server Driver para JDBC y SQLJ″
- ″Conectar con una fuente de datos utilizando la interfaz DriverManager con el controlador JDBC de DB2 de tipo 2″

Diferencia en los códigos de error y en los SQLSTATE devueltos para errores de controlador

A diferencia de los demás controladores, el IBM Data Server Driver para JDBC y SQLJ no utiliza códigos de SQL (SQLCODE) ni estados de SQL (SQLSTATE) existentes para errores internos, . Consulte ″Códigos de error emitidos por IBM Data Server Driver para JDBC y SQLJ″ y ″Estados de SQL emitidos por IBM Data Server Driver para JDBC y SQLJ″.

El controlador JDBC/SQLJ para z/OS devuelve ODBC SQLSTATE cuando se producen errores internos.

Volumen devuelto de texto de mensajes de error

Con el IBM Data Server Driver para JDBC y SQLJ, cuando ejecuta SQLException.getMessage(), no se devuelve texto de mensajes con formato a menos que la propiedad `retrieveMessagesFromServerOnGetMessage` tenga el valor `true`.

Cuando utiliza el controlador JDBC de DB2 de tipo 2, la ejecución de SQLException.getMessage() devuelve un texto de mensaje con formato.

Mecanismos de seguridad

Los controladores JDBC tienen diversos mecanismos de seguridad.

Para obtener información sobre los mecanismos de seguridad de IBM Data Server Driver para JDBC y SQLJ, consulte ″Seguridad cuando se utiliza IBM Data Server Driver para JDBC y SQLJ″.

Para obtener información sobre los mecanismos de seguridad para el controlador JDBC de DB2 de tipo 2, consulte ″Seguridad cuando se utiliza el controlador JDBC de DB2 de tipo 2″.

Soporte para conexiones de sólo lectura

Con el IBM Data Server Driver para JDBC y SQLJ, puede hacer que una conexión sea de sólo lectura mediante la propiedad `readOnly` de un objeto Connection o DataSource.

El controlador JDBC de DB2 de tipo 2 utiliza el valor Connection.setReadOnly cuando determina si es necesario establecer una conexión como de sólo lectura. No obstante, la especificación Connection.setReadOnly(true) no asegura que la conexión sea de sólo lectura.

Resultados devueltos por ResultSet.getString para una columna BIT DATA

El IBM Data Server Driver para JDBC y SQLJ devuelve datos mediante una llamada ResultSet.getString para una columna CHAR FOR BIT DATA o VARCHAR FOR BIT DATA en forma de serie hexadecimal en minúsculas.

El controlador JDBC de DB2 de tipo 2 devuelve los datos en forma de serie de caracteres hexadecimales en mayúsculas.

Resultado de una llamada executeUpdate que no afecta a ninguna fila

El IBM Data Server Driver para JDBC y SQLJ genera un SQLWarning cuando una llamada executeUpdate no afecta a ninguna fila.

El controlador JDBC de DB2 de tipo 2 no genera un SQLWarning.

Resultado de una llamada getDate o getTime para una columna TIMESTAMP

El IBM Data Server Driver para JDBC y SQLJ no genera un SQLWarning cuando se efectúa una llamada getDate o getTime para una columna TIMESTAMP.

El controlador JDBC de DB2 de tipo 2 genera un SQLWarning cuando se realiza una llamada getDate o getTime en relación a una columna TIMESTAMP.

Emisión de una excepción para PreparedStatement.setXXXStream con una discrepancia de longitud

Cuando se utilizan los métodos PreparedStatement.setBinaryStream, PreparedStatement.setCharacterStream o PreparedStatement.setUnicodeStream, el valor del parámetro *length* debe coincidir con el número de bytes de la corriente de entrada.

Si los números de bytes no coinciden, el IBM Data Server Driver para JDBC y SQLJ no emite una excepción hasta que se ejecuta el método PreparedStatement.executeUpdate posterior. Por lo tanto, para el IBM Data Server Driver para JDBC y SQLJ se pueden enviar algunos datos al servidor cuando las longitudes no coinciden. El servidor trunca o rellena estos datos. La aplicación llamadora tiene que emitir una petición de retrotracción para deshacer las actualizaciones de base de datos que incluyen los datos truncados o rellenados.

El controlador JDBC de DB2 de tipo 2 emite una excepción después de la ejecución del método PreparedStatement.setBinaryStream, PreparedStatement.setCharacterStream o PreparedStatement.setUnicodeStream.

Correlaciones por omisión para PreparedStatement.setXXXStream

Con el IBM Data Server Driver para JDBC y SQLJ, cuando se utilizan los métodos PreparedStatement.setBinaryStream, PreparedStatement.setCharacterStream o PreparedStatement.setUnicodeStream, y no se dispone de información sobre el tipo de datos de la columna de destino, los datos de entrada se correlacionan con un tipo de datos BLOB o CLOB.

Para el controlador JDBC de DB2 de tipo 2, los datos de entrada se correlacionan con un tipo de datos VARCHAR FOR BIT DATA o VARCHAR.

Cómo se realiza la conversión de caracteres

Cuando se transfieren datos de tipo carácter entre un cliente y un servidor, los datos se tienen que convertir a un formato que el receptor pueda procesar.

Para IBM Data Server Driver para JDBC y SQLJ, los datos de tipo carácter que se envían desde la fuente de datos al cliente se convierten utilizando convertidores internos de caracteres de Java. Las conversiones que soporta el IBM Data Server Driver para JDBC y SQLJ se limitan a las soportadas por la implementación de JRE subyacente.

Un cliente de IBM Data Server Driver para JDBC y SQLJ que utiliza la conectividad de tipo 4 envía datos a la fuente de datos utilizando la codificación Unicode UTF-8.

Para el controlador JDBC de DB2 de tipo 2, se pueden realizar conversiones de caracteres si las conversiones son compatibles con el servidor DB2.

Esos controladores utilizan información de CCSID procedente de la fuente de datos, si está disponible. Los controladores convierten los datos de los parámetros de entrada al CCSID de la fuente de datos antes de enviar los datos. Si la información del CCSID de destino no está disponible, los controladores envían los datos utilizando la codificación Unicode UTF-8.

Conversiones de tipos de datos implícitas o explícitas para parámetros de entrada

Si ejecuta un método PreparedStatement.setXXX, y el tipo de datos resultante del método setXXX no coincide con el tipo de datos de la columna de tabla a la que está asignado el valor de parámetro, el controlador devuelve un error a menos que se produzca una conversión del tipo de datos.

Con el IBM Data Server Driver para JDBC y SQLJ, se produce implícitamente una conversión al tipo de datos correcto de SQL si se conoce el tipo de datos de destino y las propiedades de conexión `deferPrepares` y `sendDataAsIs` están establecidas en `false`. En este caso, los valores implícitos alteran temporalmente los posibles valores explícitos de la llamada setXXX. Si la propiedad de conexión `deferPrepares` o `sendDataAsIs` se establece en `true`, debe utilizar el método PreparedStatement.setObject para convertir el parámetro al tipo de datos correcto de SQL.

Para el controlador JDBC de DB2 de tipo 2, si el tipo de datos de un parámetro no coincide con su tipo de datos SQL por omisión, debe utilizar el método PreparedStatement.setObject para convertir el parámetro al tipo de datos SQL correcto.

Soporte para las conversiones del tipo de datos String a BINARY para parámetros de entrada

El IBM Data Server Driver para JDBC y SQLJ no da soporte a las llamadas PreparedStatement.setObject del formato siguiente cuando *x* es un objeto de tipo `String`:

```
setObject(ÍndiceParámetros,
x, java.sqlTypes.BINARY)
```

El controlador JDBC de DB2 de tipo 2 soporta llamadas de este tipo. El controlador interpreta el valor de *x* como una serie de caracteres hexadecimales.

Resultado de PreparedStatement.setObject cuando existe una discrepancia de escala decimal

Con el IBM Data Server Driver para JDBC y SQLJ, si invoca PreparedStatement.setObject con un parámetro de entrada decimal y la escala del parámetro de entrada es mayor que la escala de la columna de destino, el controlador trunca los dígitos de cola del valor de entrada antes de asignar el valor a la columna.

El controlador JDBC de DB2 de tipo 2 redondea los dígitos de cola del valor de entrada antes de asignar el valor a la columna.

Rango válido para el parámetro de escala ResultSet.getBigDecimal

El formato obsoleto de ResultSet.getBigDecimal tiene un parámetro de *escala* como segundo parámetro. El IBM Data Server Driver para JDBC y SQLJ permite un rango de 0 a 32 para el parámetro de escala.

El controlador JDBC de DB2 de tipo 2 permite un rango de -1 a 32.

Soporte de conversiones del tipo de datos java.lang.Character para los parámetros de entrada

Para el formato siguiente de PreparedStatement.setObject, IBM Data Server Driver para JDBC y SQLJ da soporte a las correlaciones entre tipos de datos estándares de los objetos de Java con tipos de datos JDBC cuando convierte *x* a un tipo de datos JDBC:

```
setObject(ÍndiceParámetros, x)
```

El controlador JDBC de DB2 de tipo 2 es compatible con la correlación no estándar de *x* desde `java.lang.Character` a CHAR.

Soporte de ResultSet.getBinaryStream para una columna de caracteres

El IBM Data Server Driver para JDBC y SQLJ soporta `ResultSet.getBinaryStream` con un argumento que representa una columna de caracteres únicamente si la columna tiene el atributo FOR BIT DATA.

Para el controlador JDBC de DB2 de tipo 2, si el argumento ResultSet.getBinaryStream es una columna de caracteres, no es necesario que esa columna tenga el atributo FOR BIT DATA.

Datos devueltos por ResultSet.getBinaryStream para una columna binaria

Con el IBM Data Server Driver para JDBC y SQLJ, cuando se ejecuta ResultSet.getBinaryStream para una columna binaria, los datos devueltos están en forma de pares de dígitos hexadecimales en minúsculas.

Para el controlador JDBC de DB2 de tipo 2, cuando ejecuta ResultSet.getBinaryStream sobre una columna binaria, los datos devueltos son pares de dígitos hexadecimales, en mayúsculas.

Resultado de utilizar setObject con un tipo de entrada Boolean y un tipo de destino CHAR

Con el IBM Data Server Driver para JDBC y SQLJ, cuando ejecuta `PreparedStatement.setObject(`*`ÍndiceParámetros`*`,`*`x`*`,CHAR)`, y *x* es de tipo Boolean, se inserta el valor ″0″ o ″1″ en la columna de tabla.

Con el controlador JDBC de DB2 de tipo 2, se inserta la serie ″false″ o ″true″ en la columna de la tabla. La longitud de la columna de tabla debe ser 5 como mínimo.

Resultado de utilizar getBoolean para recuperar un valor de una columna de tipo CHAR

Con el IBM Data Server Driver para JDBC y SQLJ, cuando se ejecuta ResultSet.getBoolean o CallableStatement.getBoolean para recuperar un valor de tipo Boolean de una columna de tipo CHAR y la columna contiene el valor ″false″ o ″0″, se devuelve el valor `false`. Si la columna contiene cualquier otro valor, se devuelve `true`.

Para el controlador JDBC de DB2 de tipo 2, cuando ejecuta ResultSet.getBoolean o CallableStatement.getBoolean para recuperar un valor booleano de una columna CHAR, y la columna contiene el valor ″true″ o ″1″, se devuelve el valor `true`. Si la columna contiene cualquier otro valor, se devuelve `false`.

Resultado de ejecutar ResultSet.next() sobre un cursor cerrado

Con el IBM Data Server Driver para JDBC y SQLJ, cuando ejecuta ResultSet.next() para un cursor cerrado, se emite una excepción SQLException. Esto se ajusta a los estándares de JDBC.

Para el controlador JDBC de DB2 de tipo 2, cuando ejecuta ResultSet.next() sobre un cursor cerrado, se devuelve el valor `false` y no se emite ninguna excepción.

Resultado de especificar argumentos nulos en llamadas DatabaseMetaData

Con el IBM Data Server Driver para JDBC y SQLJ, se puede especificar `null` para un argumento de una llamada al método DatabaseMetaData únicamente si la especificación de JDBC indica que se admite `null`. De lo contrario, se emite una excepción.

Para el controlador JDBC de DB2 de tipo 2, `null` significa que el argumento no se utiliza para restringir la búsqueda.

Conversión de los argumentos de métodos a mayúsculas

El IBM Data Server Driver para JDBC y SQLJ no convierte a mayúsculas los argumentos de las llamadas de método.

El controlador JDBC de DB2 de tipo 2 convierte a mayúsculas el argumento de una llamada a Statement.setCursorName. Para evitar que el nombre del cursor se convierta a mayúsculas, delimite el nombre del cursor con el par de caracteres `\"`. Por ejemplo:

```
Statement.setCursorName("\"micursor\"");
```

Soporte para cláusulas de escape de indicación horaria

El IBM Data Server Driver para JDBC y SQLJ soporta el formato estándar de una cláusula de escape para `TIME`:

```
{t 'hh:mm:ss'}
```

Además del formato estándar, el controlador JDBC de DB2 de tipo 2 puede utilizar el formato siguiente de una cláusula de escape `TIME`:

```
{ts 'hh:mm:ss'}
```

Inclusión de una sentencia CALL en un lote de sentencias

El IBM Data Server Driver para JDBC y SQLJ soporta el uso de sentencias CALL en un lote de sentencias.

El controlador JDBC de DB2 de tipo 2 no soporta sentencias CALL en un lote de sentencias.

Eliminación de caracteres sobrantes en el texto de una sentencia de SQL

IBM Data Server Driver para JDBC y SQLJ no elimina los caracteres de espacio en blanco, tales como espacios, tabulaciones y caracteres de línea nueva, en el texto de la sentencia de SQL antes de pasar ese texto a la fuente de datos.

El controlador JDBC de DB2 de tipo 2 elimina los caracteres de espacio en blanco en el texto de la sentencia de SQL antes de pasar ese texto a la fuente de datos.

Resultado de ejecutar PreparedStatement.executeBatch

Cuando se ejecuta una sentencia PreparedStatement.executeBatch bajo el IBM Data Server Driver para JDBC y SQLJ, el controlador devuelve una matriz de números de actualizaciones `int`. Cada elemento de la matriz contiene el número de filas actualizadas por una sentencia del lote de sentencias.

Cuando se ejecuta una sentencia PreparedStatement.executeBatch bajo el controlador JDBC de DB2 de tipo 2, el controlador no puede determinar las cuentas de actualización, de manera que devuelve -3 para cada cuenta de actualización.

Soporte para SQL compuesto

El IBM Data Server Driver para JDBC y SQLJ no soporta bloques de SQL compuesto.

El código SQL compuesto permite que se agrupen varias sentencias de SQL en un solo bloque ejecutable. Por ejemplo:

```
EXEC SQL BEGIN COMPOUND ATOMIC STATIC
  UPDATE ACCOUNTS SET ABALANCE = ABALANCE + :delta
    WHERE AID = :aid;
  UPDATE TELLERS SET TBALANCE = TBALANCE + :delta
    WHERE TID = :tid;
  INSERT INTO TELLERS (TID, BID, TBALANCE) VALUES (:i, :branch_id, 0);
  COMMIT;
END COMPOUND;
```

El controlador JDBC de DB2 de tipo 2 es compatible con la ejecución de bloques de SQL compuesto con PreparedStatement.executeUpdate o Statement.executeUpdate.

Resultado de no definir un parámetro en una actualización por lotes

El IBM Data Server Driver para JDBC y SQLJ emite una excepción después de una llamada PreparedStatement.addBatch si un parámetro no está definido.

El controlador JDBC de DB2 de tipo 2 emite una excepción después de la llamada a PreparedStatement.executeBatch si no está definido un parámetro para cualquiera de las sentencias del lote de sentencias.

Posibilidad de llamar a procedimientos almacenados no catalogados

El controlador IBM Data Server Driver para JDBC y SQLJ no le permite invocar procedimientos almacenados que no están definidos en el catálogo DB2.

El controlador JDBC de DB2 de tipo 2 permite llamar a procedimientos almacenados que no estén definidos en el catálogo de DB2.

Especificación de tipos de datos para parámetros de procedimientos almacenados

Para el controlador IBM Data Server Driver para JDBC y SQLJ, si la fuente de datos no es compatible con la ejecución dinámica de la sentencia CALL, debe especificar los parámetros de la sentencia CALL **exactamente** tal como están especificados en la definición del procedimiento almacenado.

Por ejemplo, las fuentes de datos DB2 para z/OS no son compatibles con la ejecución dinámica de sentencias CALL. Supongamos que el primer parámetro de un procedimiento almacenado en un servidor DB2 para z/OS esté definido de este modo en la sentencia CREATE PROCEDURE:

```
OUT PARM1 DECIMAL(3,0)
```

En la aplicación solicitante, una sentencia como cs.registerOutParameter(1, Types.DECIMAL) no es correcta. Debe utilizar el formato del método registerOutParameter que especifica la escala, así como el tipo de datos: cs.registerOutParameter (1, Types.DECIMAL, 0).

El controlador JDBC de DB2 de tipo 2 no necesita que los tipos de datos del parámetro de una aplicación solicitante coincidan con los tipos de datos de la sentencia CREATE PROCEDURE.

Ejemplos de valores de ResultSetMetaData.getColumnName y ResultSetMetaData.getColumnLabel

Para el IBM Data Server Driver para JDBC y SQLJ Versión 4.0 y posterior, los valores devueltos para ResultSetMetaData.getColumnName y `ResultSetMetaData.getColumnLabel` difieren de los valores que se devuelven para controladores JDBC de versiones anteriores.

Los ejemplos siguientes muestran los valores que se devuelven para el 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 89. 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 Dynamic Server	MYLABEL	MYLABEL	MYCOL	MYLABEL
DB2 para z/OS Versión 8 o posterior, y DB2 para i5/OS Versión V5R3 y posterior	MYLABEL	MYLABEL	MYCOL	MYLABEL
DB2 para z/OS Versión 7, y DB2 para i5/OS Versión 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 de DB2 para z/OS o DB2 para i5/OS, 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 90. 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 posterior, y DB2 para i5/OS V5R3 y posterior	MYLABEL	LABELONCOL	MYCOL	MYLABEL
DB2 para z/OS Versión 7, y DB2 para i5/OS V5R2	MYLABEL	LABELONCOL	MYCOL	LABELONCOL

Ejemplo: en una fuente de datos de DB2 para z/OS o DB2 para i5/OS, 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 91. 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 para i5/OS V5R3 y posterior	MYCOL	LABELONCOL	MYCOL	MYCOL
DB2 para z/OS Versión 7, y DB2 para i5/OS V5R2	MYCOL	LABELONCOL	MYLABEL	LABELONCOL

Diferencias en SQLJ entre IBM Data Server Driver para JDBC y SQLJ y otros controladores JDBC de DB2

Existen varias diferencias entre el controlador IBM Data Server Driver para JDBC y SQLJ y los controladores JDBC más antiguos. Cuando pasa a utilizar el controlador IBM Data Server Driver para JDBC y SQLJ, necesita modificar sus programas SQLJ para tener en cuenta esas diferencias.

Importante: El controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows (controlador JDBC de DB2 de tipo 2) está en desuso. Esta información se ofrece para ayudarle a migrar sus aplicaciones al IBM Data Server Driver para JDBC y SQLJ.

El soporte de SQLJ en el IBM Data Server Driver para JDBC y SQLJ difiere del soporte de SQLJ existente en los demás controladores JDBC de DB2 en las áreas siguientes:

Errores de db2sqljcustomize y el parámetro -collection

El programa de utilidad db2sqljcustomize que forma parte del IBM Data Server Driver para JDBC y SQLJ tiene un parámetro -collection. El programa de utilidad db2profc que forma parte del controlador JDBC de DB2 de tipo 2 no tiene un parámetro -collection. Si el programa de utilidad db2sqljcustomize realiza una operación de vinculación en un servidor DB2 para z/OS y el parámetro -collection contiene algún carácter en minúsculas, db2sqljcustomize devuelve un error -4499, pues los ID de colección no pueden contener caracteres en minúsculas en DB2 para z/OS. Esta situación no puede producirse con db2profc.

Diferencias en los perfiles serializados

El controlador JDBC de DB2 de tipo 2 y el IBM Data Server Driver para JDBC y SQLJ producen código binario diferente cuando se ejecuta su programa traductor SQLJ y los programas de utilidad del personalizador SQLJ. Por tanto, las aplicaciones SQLJ que ha traducido y personalizado mediante los programas de utilidad `sqlj` y `db2profc` del controlador JDBC de DB2 de tipo 2 no se ejecutan bajo el IBM Data Server Driver para JDBC y SQLJ. ***Para poder ejecutar esas aplicaciones SQLJ bajo IBM Data Server Driver para JDBC y SQLJ, debe volver a traducir y personalizar las aplicaciones mediante los programas de utilidad `sqlj` y `db2sqljcustomize` de IBM Data Server Driver para JDBC y SQLJ.*** Debe hacer esto aunque no hubiera modificado las aplicaciones.

Soporte de la sentencia VALUES de SQL

El controlador JDBC de DB2 de tipo 2 permite la utilización de la sentencia VALUES de SQL en una cláusula de sentencia SQLJ, pero no ocurre así con IBM Data Server Driver para JDBC y SQLJ. Por tanto, es necesario que modifique las aplicaciones SQLJ que incluyan sentencias VALUES.

Ejemplo: suponga que un programa SQLJ contiene la sentencia siguiente:

```
#sql [ctxt] hv = {VALUES (MY_ROUTINE(1))};
```

Para el IBM Data Server Driver para JDBC y SQLJ, es necesario que cambie esa sentencia por lo siguiente:

```
#sql [ctxt] {SELECT MY_ROUTINE(1) INTO :hv FROM SYSIBM.SYSDUMMY1};
```

Soporte de sentencias de SQL compuesto

El controlador JDBC de DB2 de tipo 2 permite la utilización de sentencias de SQL compuesto en una cláusula de sentencia SQLJ, pero no ocurre así con IBM Data Server Driver para JDBC y SQLJ. Por tanto, es necesario que modifique las aplicaciones SQLJ que tengan sentencias de SQLJ donde se incluyan las especificaciones BEGIN COMPOUND y END COMPOUND. Si utiliza sentencias compuestas para efectuar actualizaciones de proceso por lotes, en su lugar puede

utilizar las interfaces de programación de SQLJ para las actualizaciones de proceso por lotes.

Diferencia en las técnicas de conexión

Las técnicas de conexión que están disponibles, y los nombres de controlador y los URL que se utilizan para esas técnicas de conexión, varían de un controlador a otro. Consulte ″Conexión con una fuente de datos utilizando SQLJ″ para obtener más información.

Soporte para iteradores desplazables y actualizables

SQLJ con el IBM Data Server Driver para JDBC y SQLJ soporta iteradores desplazables y actualizables.

El controlador JDBC de DB2 de tipo 2 para Linux, UNIX y Windows (controlador JDBC de DB2 de tipo 2) es compatible con cursores desplazables, pero no con iteradores actualizables.

Ejecución dinámica de sentencias de SQL bajo WebSphere Application Server

Para WebSphere Application Server Versión 5.0.1 y superior, si personaliza el programa SQLJ, las sentencias de SQL se ejecutan de manera estática.

Los nombres alternativos para db2sqljcustomize y db2sqljprint no están soportados

El controlador DB2 JDBC Tipo 2 utilizaba originariamente el nombre db2profc para el mandato de personalizador de perfil SQLJ, y el nombre db2profp para el mandato de impresora de perfil SQLJ. Para IBM Data Server Driver para JDBC y SQLJ, el mandato de personalizador de perfil SQLJ se denomina db2sqljcustomize, y el mandato de impresora de perfil SQLJ se denomina db2sqljprint. En releases anteriores de DB2 Database para Linux, UNIX y Windows, db2profc se aceptaba como nombre alternativo de db2sqljcustomize, y db2profp se aceptaba como nombre alternativo de db2sqljprint. Estos nombres alternativos ya no se aceptan.

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 el IBM Data Server Driver para JDBC y SQLJ.

Valores recuperados para caracteres de sustitución DBCS

Cuando recupera un carácter de sustitución DBCS, tal como X'FCFC' de la página de códigos Cp943, de una tabla de base de datos, el valor devuelto es diferente dependiendo de si está utilizando un SDK de Java de IBM o un SDK de Java de Sun.

Para un SDK de Java de Sun, 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 pueden trabajar con más páginas de códigos DBCS que los SDK de Sun para Java. Por tanto, si obtiene errores debido a la utilización de páginas de códigos no permitidas con un SDK de Sun para Java, puede probar utilizar un SDK de IBM para Java.

Necesidad de cifrado para el SDK de IBM para Java

Los SDK de IBM para Java son compatibles con el cifrado de 256 bits, pero no así los SDK de Sun para Java. 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 el 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 el 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 explicatorio
- El código de error
- El estado de SQL (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, el IBM Data Server Driver para JDBC y SQLJ emite los códigos de error siguientes:

Tabla 92. Códigos de error emitidos por el 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
+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 en la vinculación o la personalización de una aplicación SQLJ. **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.	

Tabla 92. Códigos de error emitidos por el IBM Data Server Driver para JDBC y SQLJ (continuación)

Código de error	Texto del mensaje y explicación	SQLSTATE
+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.	
+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.	
-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 92. Códigos de error emitidos por el 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, el 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.	
-4211	Tiempo de espera excedido al obtener un objeto de la agrupación.	
-4212	Miembro de Sysplex no disponible.	
-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.	

Tabla 92. Códigos de error emitidos por el IBM Data Server Driver para JDBC y SQLJ (continuación)

Código de error	Texto del mensaje y explicación	SQLSTATE
-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.	
-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 en la vinculación o la personalización de una aplicación SQLJ. **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.	
-4296	Excepción *clase-excepción*: se produjo un error al abrir el socket para el servidor *servidor* en el puerto *puerto* con el mensaje *mensaje*. **Explicación:** el controlador no pudo establecer una conexión. **Respuesta de usuario:** verifique que el nombre o la dirección del sistema principal y el número de puerto del servidor son correctos. Verifique también que éste esté iniciado. En el caso de una conexión con una fuente de datos de DB2 Database para Linux, UNIX y Windows, verifique que el parámetro de configuración svcename está configurado de forma adecuada.	08001

Tabla 92. Códigos de error emitidos por el IBM Data Server Driver para JDBC y SQLJ (continuación)

Código de error	Texto del mensaje y explicación	SQLSTATE
-4297	Se ha producido un error de comunicación durante operaciones en el socket asociado a la conexión, la corriente de entrada del socket o la corriente de salida del socket. Ubicación del error: *función*. Mensaje: *mensaje*.	08001
-4298	El gestor de bases de datos no puede aceptar nuevas peticiones, ha concluido todas las peticiones en curso o ha terminado la petición actual debido a que se detectaron condiciones de error inesperadas en el sistema de destino.	58009
-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.	42815
-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.	

Tabla 92. Códigos de error emitidos por el IBM Data Server Driver para JDBC y SQLJ (continuación)

Código de error	Texto del mensaje y explicación	SQLSTATE
-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.	
-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.	

Tabla 92. Códigos de error emitidos por el IBM Data Server Driver para JDBC y SQLJ (continuación)

Código de error	Texto del mensaje y explicación	SQLSTATE
-4498	Ha fallado una conexión pero se ha restablecido. Dirección IP o nombre de sistema principal: *nombre-sistema-principal*, número de puerto o nombre de servicio: *puerto*, indicador de modificación de registro especial: *rc*. **Explicación:** *nombre-sistema-principal* y *puerto* indican la fuente de datos en la que se restablece la conexión. *rc* indica si las sentencias SQL que establecen valores de registro especial se ejecutaron de nuevo: **1** Las sentencias SQL que establecen valores de registro especial se ejecutaron de nuevo. **2** Las sentencias SQL que establecen valores de registro especial podrían no haberse ejecutado de nuevo. 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.	
-4499	Se ha producido un error grave que ha dado como resultado una desconexión. La conexión existente ha pasado a estar inutilizable. **Explicación:** Una causa posible es que un error de red haya provocado la desconexión de un socket.	
-30108	Excepción de redireccionamiento del cliente para Sysplex.	08506
-99999	El 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 el 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 93. 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
	02501	La posición del cursor no es válida para una operación FETCH sobre la fila actual.

Tabla 93. Estados de SQL devueltos por IBM Data Server Driver para JDBC y SQLJ (continuación)

Clase de estado de SQL	SQLSTATE	Descripción
	02506	Error tolerable
08*xxx*		Excepción de conexión
	08003	No existe una conexión
	08004	El servidor de aplicaciones ha rechazado el establecimiento de la conexión
	08506	Excepción de redireccionamiento del cliente
0A*xxx*		Función no admitida
	0A502	La acción u operación no está permitida para esta instancia de base de datos
22*xxx*		Excepción de datos
	22007	La representación de caracteres de un valor de fecha y hora no es válida
	22021	Un carácter no pertenece al juego de caracteres codificados
23*xxx*		Violación de restricción
	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
	24501	El cursor identificado no está abierto
28*xxx*		Excepción de autorización
	28000	Nombre de autorización no válido.
2D*xxx*		Terminación de transacción no válida
	2D521	Las operaciones COMMIT o ROLLBACK de SQL no son válidas en el entorno operativo actual.
34*xxx*		Nombre de cursor no válido
	34000	El nombre del cursor no es válido.
3B*xxx*		Punto de salvaguarda no válido
	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 regla de acceso
	42601	Un carácter, distintivo o cláusula no es válido o se ha omitido
	42734	Se ha detectado un duplicado para nombre de parámetro, nombre de variable de SQL, nombre de cursor, nombre de condición o etiqueta.
	42807	La operación INSERT, UPDATE o DELETE no está permitida en este objeto
	42808	Una columna identificada en la operación de inserción o actualización no es actualizable
	42815	El tipo de datos, longitud, escala, valor o CCSID no es válido

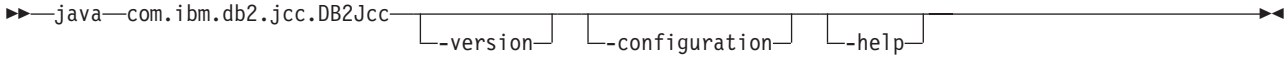
Tabla 93. Estados de SQL devueltos por IBM Data Server Driver para JDBC y SQLJ (continuación)

Clase de estado de SQL	SQLSTATE	Descripción
	42820	Una constante numérica es demasiado larga o tiene un valor que no está dentro del rango de su tipo de datos
	42968	La conexión ha fallado debido a que no existe ninguna licencia de software actual.
58*xxx*		Error del sistema
	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
	58009	La ejecución ha fallado debido a un error de protocolo de distribución que hizo que se desasignara la conversación
	58012	El proceso de vinculación con el nombre de paquete y distintivo de coherencia especificados no está activo
	58014	El mandato de DDM no está permitido
	58015	El objeto de DDM no está permitido
	58016	El parámetro de DDM no está permitido
	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 el IBM Data Server Driver para JDBC y SQLJ muestra su nombre y versión.

-configuration
Especifica que el 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.

Datos de salida de ejemplo de DB2Jcc

Los datos de salida siguientes son el resultado de invocar DB2Jcc con el parámetro -configuration.

Figura 62. Datos de salida de ejemplo de DB2Jcc

```
(myid@mymachine) /home/myusrid $ java com.ibm.db2.jcc.DB2Jcc -version
[jcc] Driver: IBM DB2 JDBC Universal Driver Architecture 3.50.137

(myid@mymachine) /home/myusrid $ java com.ibm.db2.jcc.DB2Jcc -configuration
[jcc] BEGIN TRACE_DRIVER_CONFIGURATION
[jcc] Driver: IBM DB2 JDBC Universal Driver Architecture 3.50.137
[jcc] Compatible JRE versions: { 1.4, 1.5 }
[jcc] Target server licensing restrictions: { z/OS: enabled; SQLDS: enabled; iSe
ries: enabled; DB2 for Unix/Windows: enabled; Cloudscape: enabled; Informix: ena
bled }
[jcc] Range checking enabled: true
[jcc] Bug check level: 0xff
[jcc] Default fetch size: 64
[jcc] Default isolation: 2
[jcc] Collect performance statistics: false
[jcc] No security manager detected.
[jcc] Detected local client host: lead.svl.ibm.com/9.30.10.102
[jcc] Access to package sun.io is permitted by security manager.
[jcc] JDBC 1 system property jdbc.drivers = null
[jcc] Java Runtime Environment version 1.4.2
[jcc] Java Runtime Environment vendor = IBM Corporation
[jcc] Java vendor URL = http://www.ibm.com/
[jcc] Java installation directory = /wsdb/v91/bldsupp/AIX5L64/jdk1.4.2_sr1/sh/..
/jre
[jcc] Java Virtual Machine specification version = 1.0
[jcc] Java Virtual Machine specification vendor = Sun Microsystems Inc.
[jcc] Java Virtual Machine specification name = Java Virtual Machine Specificati
on
[jcc] Java Virtual Machine implementation version = 1.4.2
[jcc] Java Virtual Machine implementation vendor = IBM Corporation
[jcc] Java Virtual Machine implementation name = Classic VM
[jcc] Java Runtime Environment specification version = 1.4
[jcc] Java Runtime Environment specification vendor = Sun Microsystems Inc.
[jcc] Java Runtime Environment specification name = Java Platform API Specificat
ion
[jcc] Java class format version number = 48.0
[jcc] Java class path = :.:/home2/myusrid/sqllib/java/db2java.zip:/lib/classes.z
ip:/home2/myusrid/sqllib/java/sqlj.zip:./test:/home2/myusrid/sqllib/java/db2jcc.
jar:/home2/myusrid/sqllib/java/db2jcc_license_cisuz.jar:...
[jcc] Java native library path = /wsdb/v91/bldsupp/AIX5L64/jdk1.4.2_sr1/sh/../jr
e/bin:/wsdb/v91/bldsupp/AIX5L64/jdk1.4.2_sr1/jre/bin/classic:/wsdb/v91/bldsupp/A
IX5L64/jdk1.4.2_sr1/jre/bin:/home2/myusrid/sqllib/lib:/local/cobol:/home2/myusri
d/sqllib/samples/c:/usr/lib
[jcc] Path of extension directory or directories = /wsdb/v91/bldsupp/AIX5L64/jdk
1.4.2_sr1/sh/../jre/lib/ext
[jcc] Operating system name = AIX
[jcc] Operating system architecture = ppc64
[jcc] Operating system version = 5.3
[jcc] File separator ("/" on UNIX) = /
[jcc] Path separator (":" on UNIX) = :
[jcc] User's account name = myusrid
[jcc] User's home directory = /home2/myusrid
[jcc] User's current working directory = /home2/myusrid
[jcc] Dumping all system properties: { java.assistive=ON, java.runtime.name=Java
(TM) 2 Runtime Environment, Standard Edition, sun.boot.library.path=/wsdb/v91/bl
dsupp/AIX5L64/jdk1.4.2_sr1/sh/../jre/bin, java.vm.version=1.4.2, java.vm.vendor=
IBM Corporation, java.vendor.url=http://www.ibm.com/, path.separator=:, java.vm.
```

```
name=Classic VM, file.encoding.pkg=sun.io, user.country=US, sun.os.patch.level=u
nknown, ... }
[jcc] Dumping all file properties: {  }
[jcc] END TRACE_DRIVER_CONFIGURATION
```

Mandatos para la preparación de programas de SQLJ

Para preparar programas SQLJ para su ejecución, debe utilizar mandatos para convertir código fuente SQLJ en código fuente Java, compilar el código fuente Java, personalizar perfiles serializados de SQLJ, y vincular paquetes DB2.

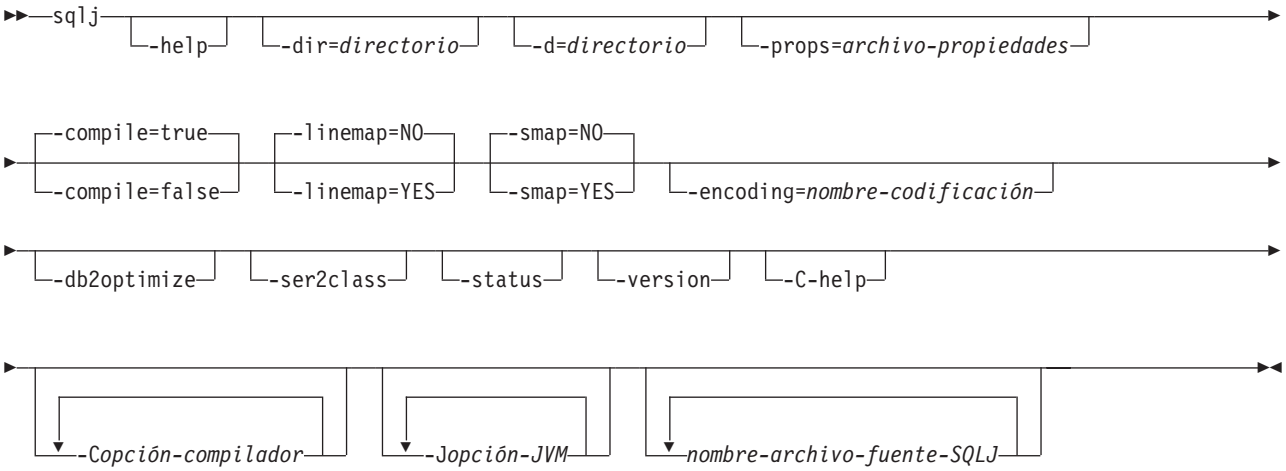
sqlj - Traductor SQLJ

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 traductor 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 Java
- file2.sqlj, que está en el paquete 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 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 Java
- file2.sqlj, que está en el paquete 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 traductor SQLJ debe compilar el fuente de Java generado 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 traductor 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 traductor 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 traductor SQLJ. El archivo está en el esquema de codificación Unicode UTF-8. Su formato se describe en la solicitud 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 traductor 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 traductor SQLJ crea código para una clase de contexto de conexión que está optimizado para DB2. -db2optimize optimiza el código para el contexto definido por el usuario, aunque no para el contexto por omisión.

Cuando ejecuta el traductor SQLJ con la opción -db2optimize, si sus aplicaciones utilizan JDBC 3.0 o funciones de versiones anteriores, el archivo db2jcc.jar de IBM Data Server Driver para JDBC y SQLJ debe estar incluido en la variable CLASSPATH para compilar la aplicación Java generada. Si sus aplicaciones utilizan JDBC 4.0 o funciones de versiones anteriores, el archivo db2jcc4.jar de IBM Data Server Driver para JDBC y SQLJ debe estar incluido en la variable CLASSPATH para compilar la aplicación Java generada.

-ser2class
Especifica que el traductor SQLJ convierte archivos .ser en archivos .class.

-status
Especifica que el traductor SQLJ muestra mensajes de estado durante su ejecución.

-version
Especifica que el traductor SQLJ muestra la versión del IBM Data Server Driver para JDBC y SQLJ. La información es de la forma:

```
IBM SQLJ xxxx.xxxx.xx
```

-C-help
Especifica que el traductor 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 traductor SQLJ, **a excepción** de las opciones siguientes:

-classpath
Especifica la vía de acceso a clases del usuario que van a utilizar el traductor 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 traductor SQLJ y el compilador Java buscarán definiciones de clases o interfaces. El traductor 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 traductor SQLJ genera los archivos siguientes:

- El programa fuente generado

  El archivo fuente generado se denomina *nombre-programa*.java.
- Un archivos 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 traductor 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 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.

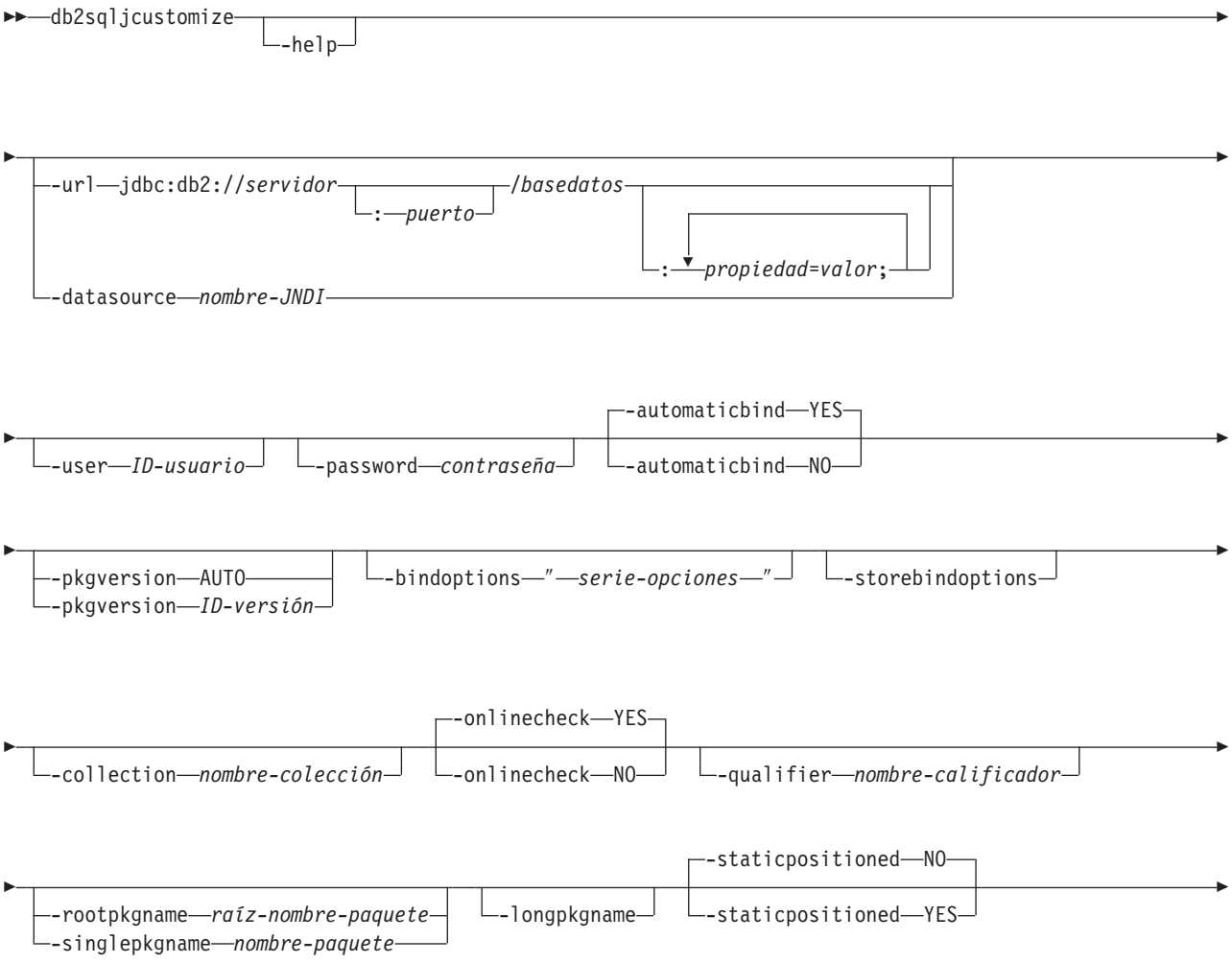
Autorización

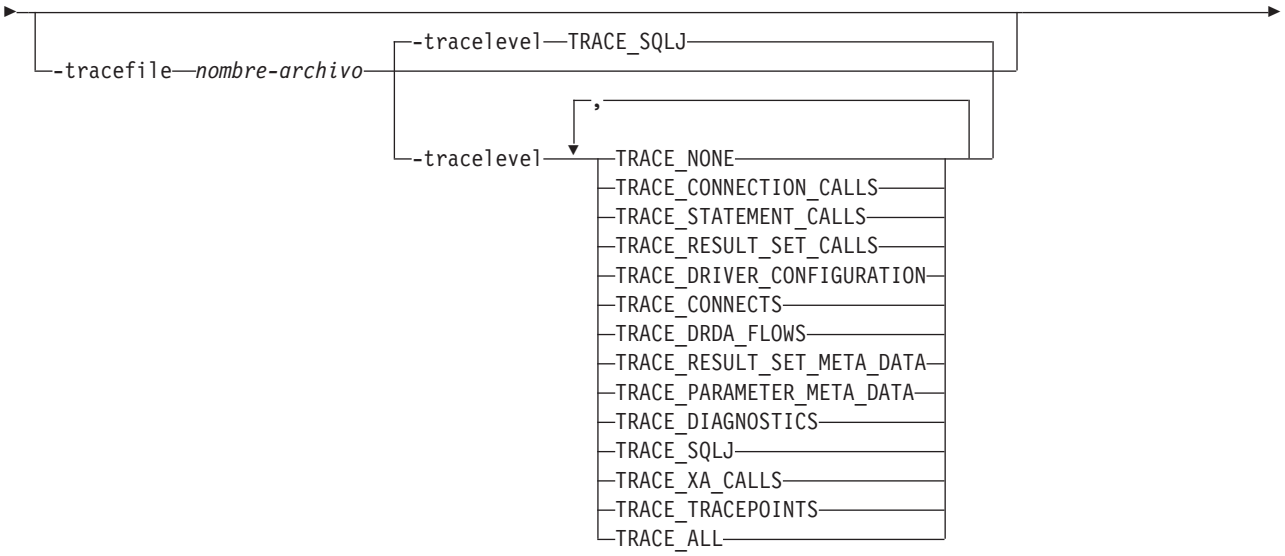
El conjunto de privilegios del proceso tiene que incluir una de las autorizaciones posibles:

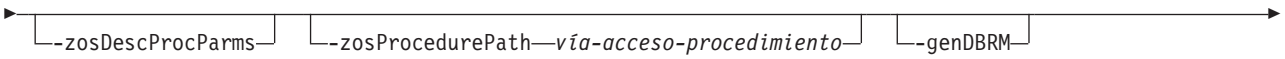
- Autorización SYSADM
- Autorización DBADM
- Si el paquete no existe, el privilegio BINDADD y uno de los privilegios siguientes:
  - Privilegio CREATEIN
  - Autorización IMPLICIT_SCHEMA sobre la base de datos si el nombre de esquema del paquete no existe
- Si el paquete existe:
  - Privilegio ALTERIN sobre el esquema
  - Privilegio BIND sobre el paquete

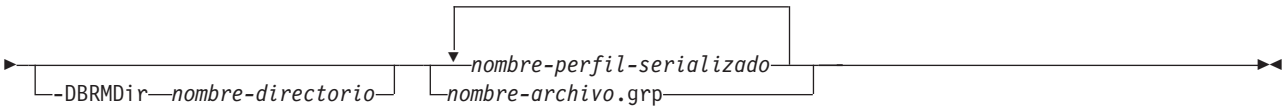
El usuario también necesita todos los privilegios necesarios para compilar las sentencias de SQL estáticas de la aplicación. Los privilegios otorgados a grupos no se utilizan para la comprobación de autorizaciones de sentencias estáticas. Si el usuario tiene autorización SYSADM, pero no tiene privilegios explícitos para ejecutar la vinculación, el gestor de bases de datos DB2 otorga automáticamente autorización DBADM explícita.

Sintaxis del mandato

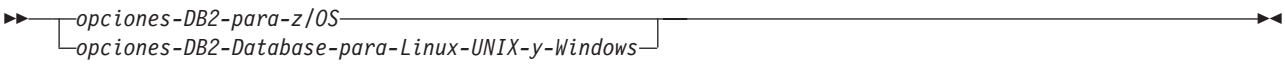




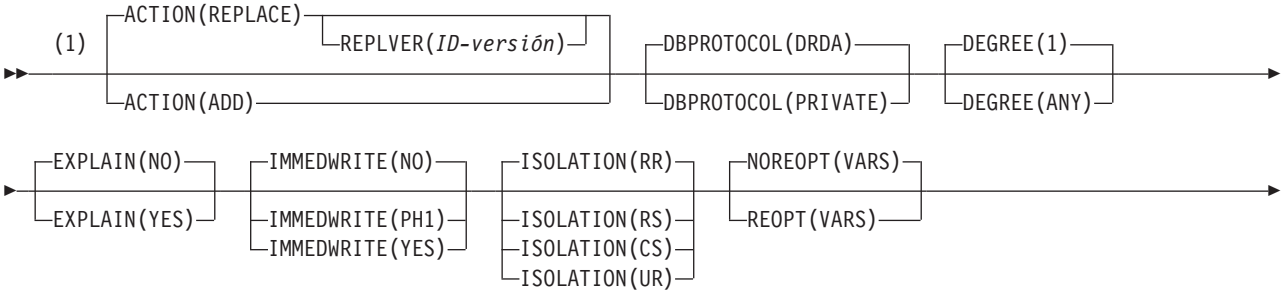


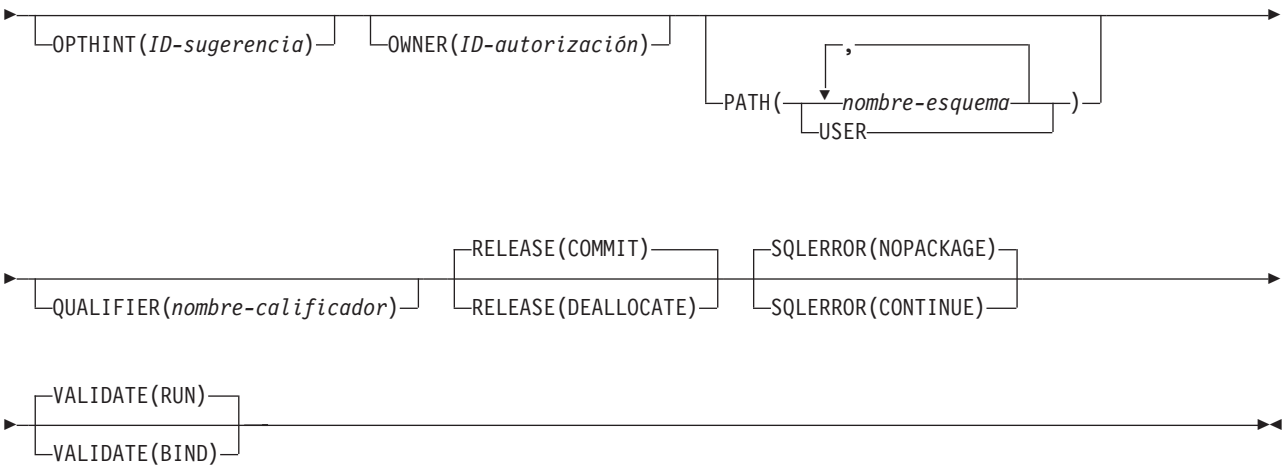


serie-opciones:



DB2 para z/OS opciones:

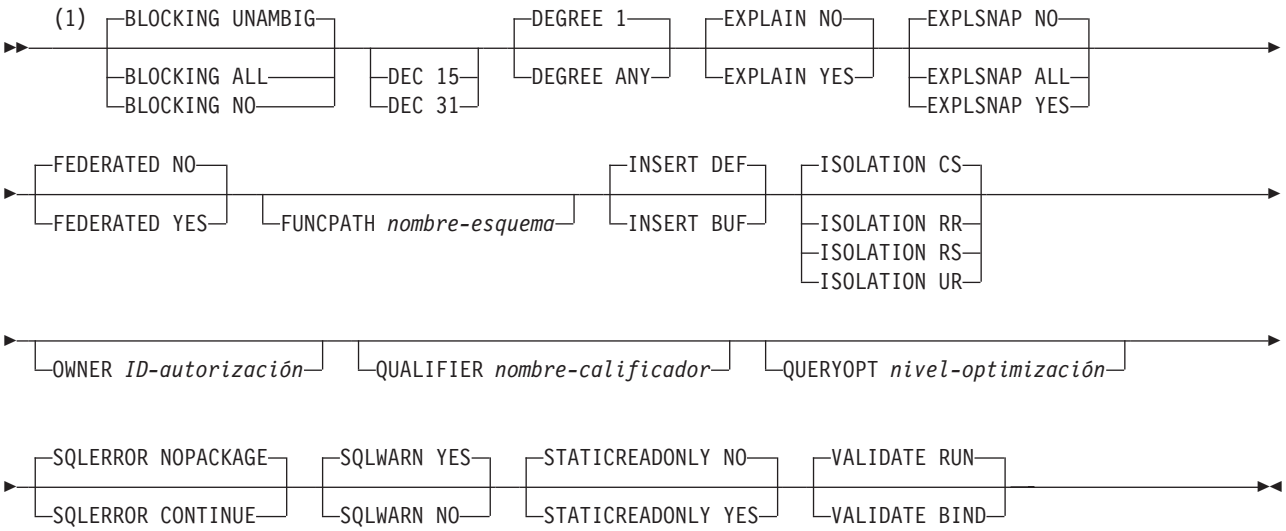




Notas:

1 Estas opciones se pueden especificar en cualquier orden.

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, el valor de *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 el IBM Data Server Driver para conectividad JDBC y SQLJ 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 la señal de coherencia y no en el nombre de la versión. Para generar automáticamente un nombre de versión basado en la señal 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 nombre 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 94 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 94. 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 95.

La Tabla 95 muestra los valores de la porción *Aislamientopqt* de un nombre de paquete generado por db2sqljcustomize.

Tabla 95. 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 DB2 para z/OS realiza una operación DESCRIBE sobre parámetros de procedimientos almacenados.

-zosDescProcParms sólo se aplica a los programas que se deben ejecutar sobre servidores de bases de datos DB2 para z/OS.

Si hay disponible información de DESCRIBE, SQLJ tiene información sobre la longitud y la precisión de parámetros INOUT y OUT, de modo que sólo asigna la cantidad de memoria necesaria para dichos parámetros. La disponibilidad de información de DESCRIBE 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 bases 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 DESCRIBE.

-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 genera módulos de solicitud de base de datos (DBRM). Estos DBRM se pueden utilizar para crear planes y 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.

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 de serie del lenguaje principal 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 tabla 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 o planes 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 o planes, deberá especificar los siguientes valores de opciones de vinculación:

ENCODING(EBCDIC)
: El IBM Data Server Driver para JDBC y SQLJ en DB2 para z/OS necesita la codificación EBCDIC para los paquetes y los planes.

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.

DBPROTOCOL(DRDA)
El protocolo privado ha quedado en desuso y, por lo tanto, se debe utilizar DBPROTOCOL(DRDA) para todas las aplicaciones. No obstante, para las aplicaciones SQLJ que utilicen nombres de tabla remotas de tres partes, se debe utilizar DBPROTOCOL(DRDA). De lo contrario, estas aplicaciones podrían fallar.

db2sqljbind - vinculador de perfiles SQLJ

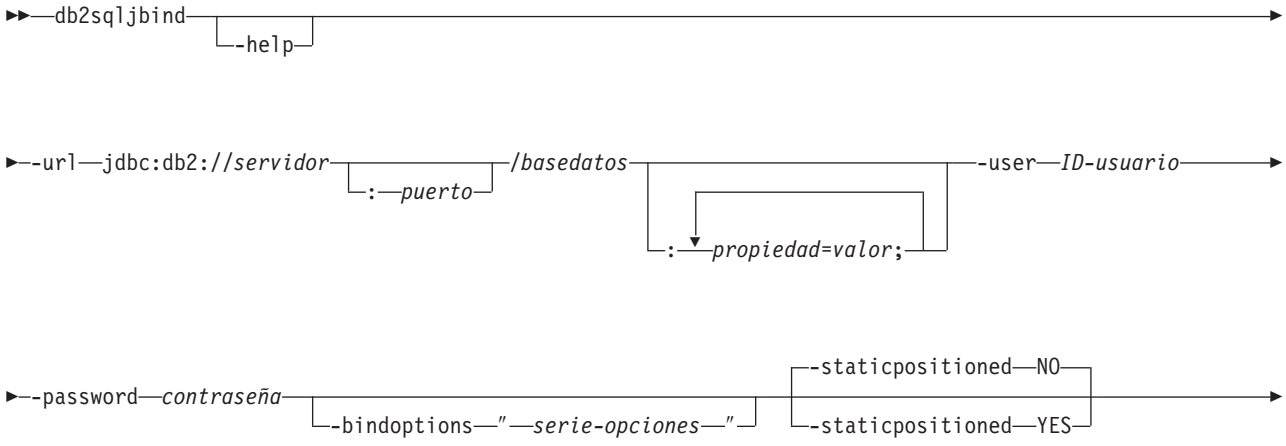
db2sqljbind vincula paquetes DB2 para un perfil serializado que previamente se ha personalizado mediante el mandato db2sqljcustomize.

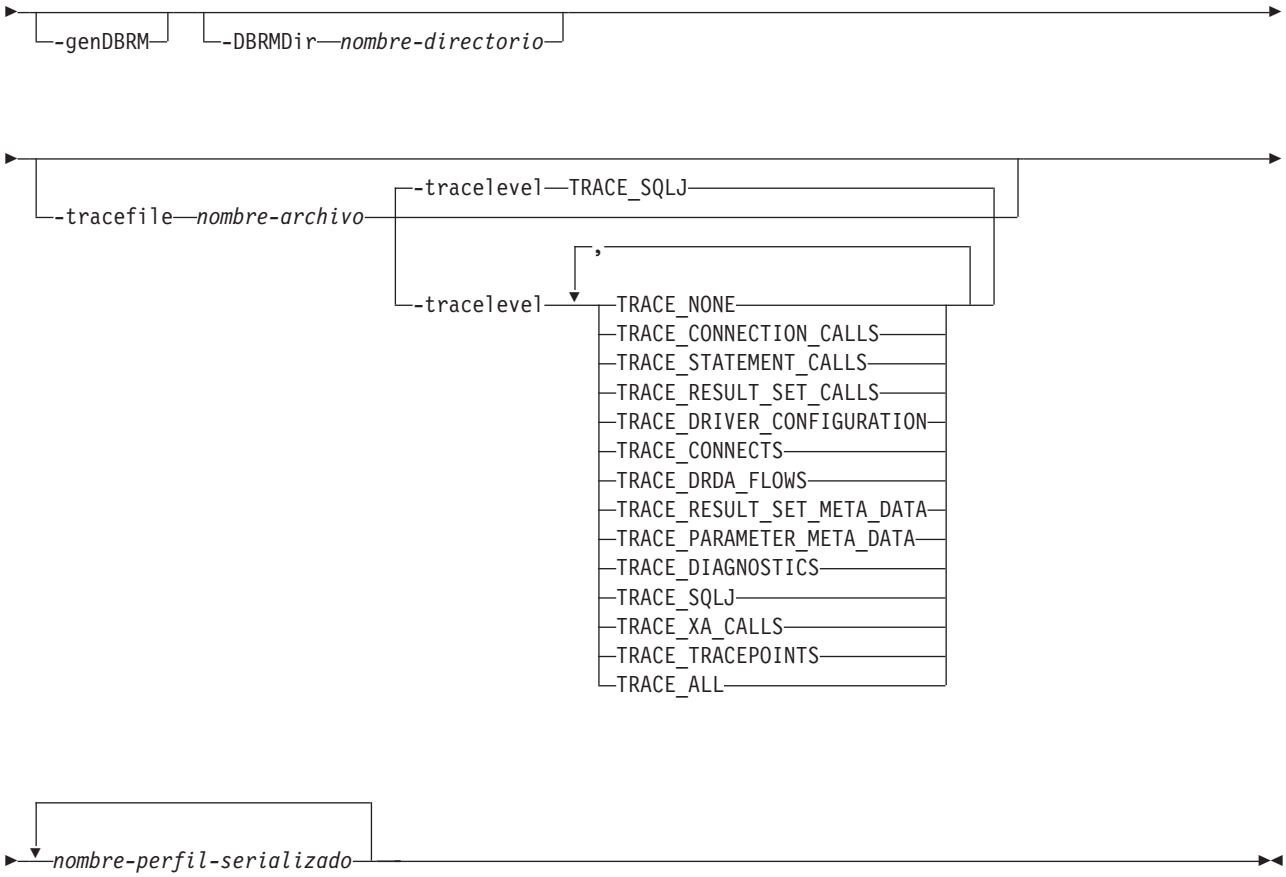
Autorización

El conjunto de privilegios del proceso tiene que incluir una de las autorizaciones posibles:
- Autorización SYSADM
- 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

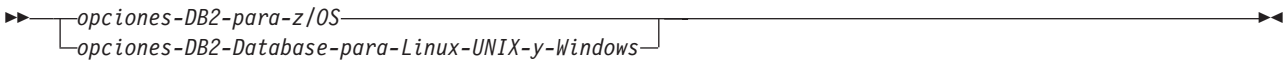
El usuario también necesita todos los privilegios necesarios para compilar las sentencias de SQL estáticas de la aplicación. Los privilegios otorgados a grupos no se utilizan para la comprobación de autorizaciones de sentencias estáticas. Si el usuario tiene autorización SYSADM, pero no tiene privilegios explícitos para ejecutar la vinculación, el gestor de bases de datos DB2 otorga automáticamente autorización DBADM explícita.

Sintaxis del mandato

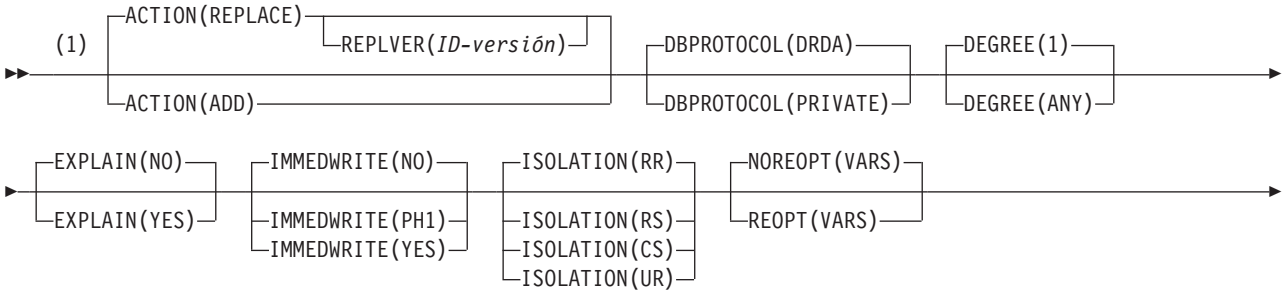


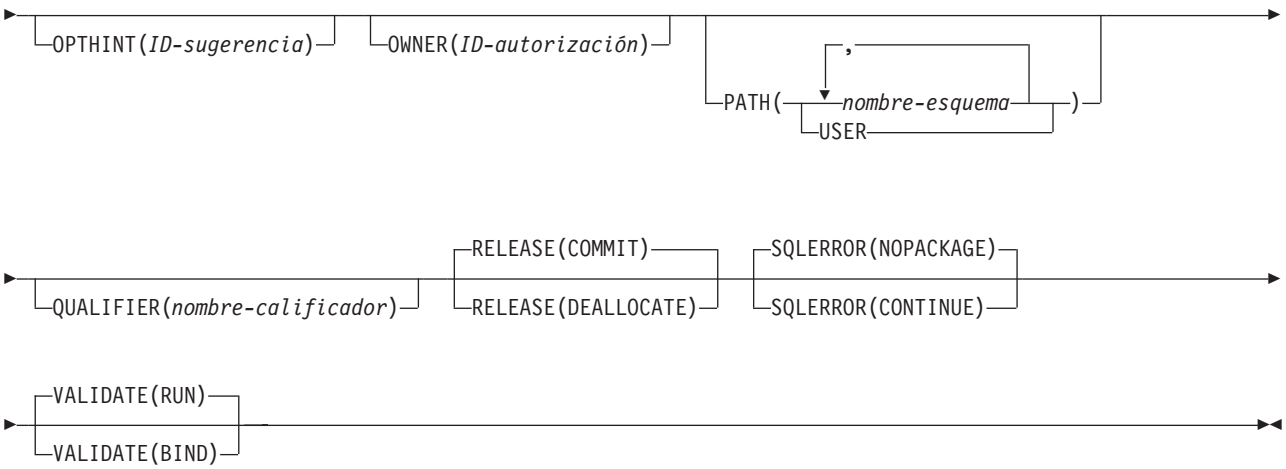


serie-opciones:



DB2 para z/OS opciones:

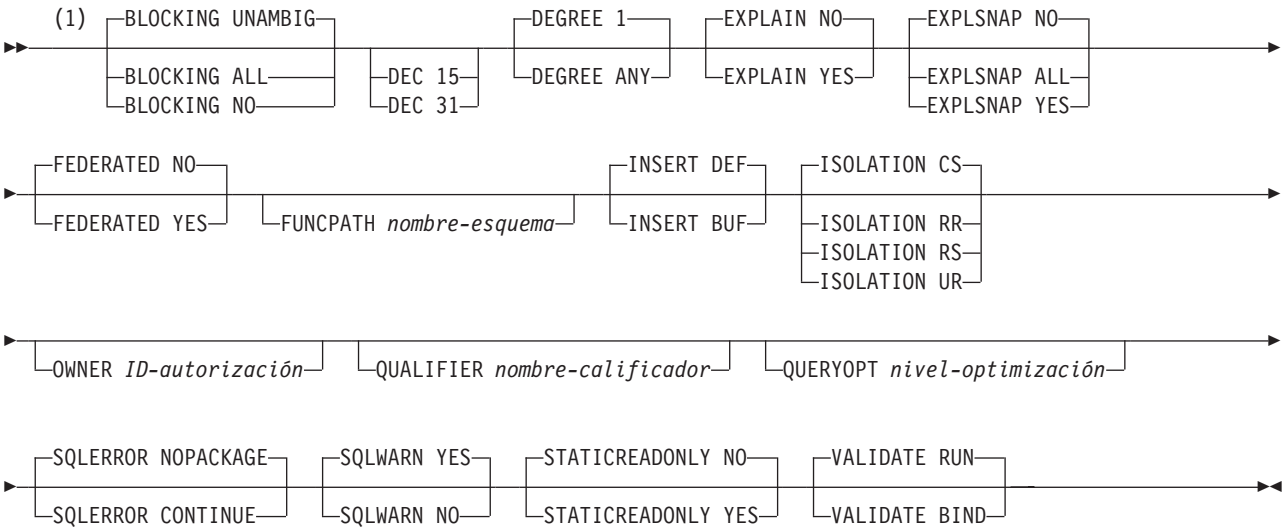




Notas:

1 Estas opciones se pueden especificar en cualquier orden.

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, el valor de *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.

-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 su programa para su ejecución en un sistema DB2 para z/OS, especifique opciones de DB2 para z/OS. Si está preparando su programa para su ejecución 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 resolver errores de declaración de iterador, es necesario especificar explícitamente STATICREADONLY NO, o declarar

iteradores como no ambiguos. Por ejemplo, si desea que un iterator sea actualizable de forma no ambigua, declare el iterador para implementar sqlj.runtime.ForUpdate. Si desea que un iterador sea de solo lectura, incluya la cláusula FOR READ ONLY en las sentencias SELECT donde se utiliza 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 que están declarados en el mismo archivo fuente que las sentencias UPDATE de posición donde se utilizan los iteradores, especifica si las sentencias UPDATE de posición se ejecutan 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 solicitud 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 se debe especificar solamente bajo la dirección del centro de 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 se debe especificar solamente bajo la dirección del centro de soporte de software de IBM.

nombre-perfil-serializado
Especifica el nombre 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_SJProfileNúmeroID.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.

Si especifica más de un nombre de perfil serializado para vincular un paquete DB2 simple de entre varios perfiles serializados, debe haber especificado los mismos nombres de perfiles serializados, en el mismo orden, cuando ejecutó db2sqljcustomize.

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 dinámicas 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.

db2sqljprint - impresora de perfiles 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 a través de las herramientas y los métodos siguientes:

- Centro de información de DB2
  - Temas (Tareas, concepto y temas de consulta)
  - Ayuda para herramientas de DB2
  - Programas de ejemplo
  - Guías de aprendizaje
- Manuales de DB2
  - Archivos PDF (descargables)
  - Archivos PDF (desde el DVD en PDF de DB2)
  - Manuales en copia impresa
- Ayuda de 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 en línea IBM Redbooks en 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 responder directamente a cada uno. Proporcione ejemplos específicos siempre que sea posible de manera que podamos comprender mejor sus problemas. Si realiza comentarios sobre un tema o archivo de ayuda determinado, incluya 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 de DB2 que no está tratado por la documentación, consulte al centro local de servicio técnico de IBM para obtener ayuda.

Biblioteca técnica de DB2 en copia impresa o en formato PDF

Las tablas siguientes describen la biblioteca de DB2 que está disponible en el Centro de publicaciones de IBM en www.ibm.com/shop/publications/order. Los manuales de DB2 Versión 9.5 en inglés en formato PDF y las versiones traducidas se pueden descargar del sitio www.ibm.com/support/docview.wss?rs=71 &uid=swg2700947.

Aunque las tablas identifican los manuales en copia impresa disponibles, puede que dichos manuales no estén disponibles en su país o región.

Tabla 96. Información técnica de DB2

Nombre	Número de documento	Copia impresa disponible
Consulta de las API administrativas	SC11-3505-00	Sí
Rutinas y vistas administrativas	SC11-3507-00	No
Call Level Interface Guide and Reference, Volume 1	SC23-5844-00	Sí
Call Level Interface Guide and Reference, Volume 2	SC23-5845-00	Sí
Consulta de mandatos	SC11-3506-00	Sí
Data Movement Utilities Guide and Reference	SC23-5847-00	Sí
Data Recovery and High Availability Guide and Reference	SC23-5848-00	Sí
Data Servers, Databases, and Database Objects Guide	SC23-5849-00	Sí
Database Security Guide	SC23-5850-00	Sí
Desarrollo de aplicaciones ADO.NET y OLE DB	SC11-3499-00	Sí
Desarrollo de aplicaciones de SQL incorporado	SC11-3500-00	Sí
Desarrollo de aplicaciones Java	SC11-3501-00	Sí
Desarrollo de aplicaciones Perl y PHP	SC11-3502-00	No
Desarrollo de rutinas definidas por el usuario (SQL y externas)	SC11-3503-00	Sí
Iniciación al desarrollo de aplicaciones de bases de datos	GC11-3504-00	Sí
Iniciación a la instalación y administración de DB2 en Linux y Windows	GC11-3511-00	Sí
Internationalization Guide	SC23-5858-00	Sí
Consulta de mensajes, Volumen 1	GI11-7823-00	No
Consulta de mensajes, Volumen 2	GI11-7824-00	No
Guía de migración	GC11-3510-00	Sí

Tabla 96. Información técnica de DB2 (continuación)

Nombre	Número de documento	Copia impresa disponible
Net Search Extender Guía de administración y del usuario **Nota:** El contenido de este documento no está incluido en el Centro de información de DB2	SC11-3615-00	Sí
Partitioning and Clustering Guide	SC23-5860-00	Sí
Query Patroller Administration and User's Guide	SC23-8507-00	Sí
Guía rápida de iniciación para clientes IBM Data Server	GC11-3513-00	No
Guía rápida de iniciación para servidores DB2	GC11-3512-00	Sí
Spatial Extender and Geodetic Data Management Feature Guía del usuario y manual de consulta	SC11-3614-00	Sí
Consulta de SQL, Volumen 1	SC11-3508-00	Sí
Consulta de SQL, Volumen 2	SC11-3509-00	Sí
System Monitor Guide and Reference	SC23-5865-00	Sí
Text Search Guide	SC23-5866-00	Sí
Troubleshooting Guide	GI11-7857-00	No
Tuning Database Performance	SC23-5867-00	Sí
Guía de aprendizaje de Visual Explain	SC11-3518-00	No
Novedades	SC11-3517-00	Sí
Workload Manager Guide and Reference	SC23-5870-00	Sí
pureXML Guide	SC23-5871-00	Sí
XQuery Reference	SC23-5872-00	No

Tabla 97. Información técnica específica de DB2 Connect

Nombre	Número de documento	Copia impresa disponible
Guía rápida de iniciación para DB2 Connect Personal Edition	GC11-3515-00	Sí
Guía rápida de iniciación para servidores DB2 Connect	GC11-3516-00	Sí
Guía del usuario de DB2 Connect	SC11-3514-00	Sí

Tabla 98. Información técnica de Information Integration

Nombre	Número de documento	Copia impresa disponible
Information Integration: Administration Guide for Federated Systems	SC19-1020-01	Sí

Tabla 98. Información técnica de Information Integration (continuación)

Nombre	Número de documento	Copia impresa disponible
Information Integration: ASNCLP Program Reference for Replication and Event Publishing	SC19-1018-02	Sí
Information Integration: Configuration Guide for Federated Data Sources	SC19-1034-01	No
Information Integration: SQL Replication Guide and Reference	SC19-1030-01	Sí
Information Integration: Introduction to Replication and Event Publishing	SC19-1028-01	Sí

Pedido de manuales de DB2 en copia impresa

Si necesita manuales de DB2 en copia impresa, puede comprarlos en línea en varios, pero no en todos los países o regiones. Siempre puede hacer pedidos de manuales de DB2 en copia impresa a través del representante local de IBM. Recuerde que algunas publicaciones en copia software del DVD *Documentación en PDF de DB2* no están disponibles en copia impresa. Por ejemplo, no está disponible la publicación *Consulta de mensajes de DB2* en copia impresa.

Las versiones impresas de muchas de las publicaciones de DB2 disponibles en el DVD de Documentación en PDF de DB2 se pueden solicitar a IBM por una cantidad. Dependiendo desde dónde realice el pedido, podrá solicitar manuales en línea, desde el Centro de publicaciones de IBM. Si la realización de pedidos en línea no está disponible en su país o región, siempre puede hacer pedidos de manuales de DB2 en copia impresa al representante local de IBM. Tenga en cuenta que no todas las publicaciones del DVD de Documentación en PDF de DB2 están disponibles en copia impresa.

Nota: La documentación más actualizada y completa de DB2 se conserva en el Centro de información de DB2 en http://publib.boulder.ibm.com/infocenter/db2luw/v9r5.

Para hacer pedidos de manuales de DB2 en copia impresa:

- Para averiguar si puede hacer pedidos de manuales de DB2 en copia impresa en línea en su país o región, consulte el Centro de publicaciones de IBM en el sitio http://www.ibm.com/shop/publications/order. Debe seleccionar un país, región o idioma para poder acceder a la información sobre pedidos de publicaciones y, a continuación, seguir las instrucciones sobre pedidos para su localidad.
- Para hacer pedidos de manuales de DB2 en copia impresa a través del representante local de IBM:
  1. Localice la información de contacto de su representante local desde uno de los siguientes sitios Web:
     - El directorio de IBM de contactos en todo el mundo en el sitio www.ibm.com/planetwide
     - El sitio Web de publicaciones de IBM en el sitio http://www.ibm.com/shop/publications/order. Tendrá que seleccionar su país, región o idioma para acceder a la página de presentación de las publicaciones apropiadas para su localidad. Desde esta página, siga el enlace "Acerca de este sitio".

2. Cuando llame, indique que desea hacer un pedido de una publicación de DB2.
3. Proporcione al representante los títulos y números de documento de las publicaciones que desee solicitar. Si desea consultar los títulos y los números de pedido, consulte el apartado "Biblioteca técnica de DB2 en copia impresa o en formato PDF" en la página 434.

Visualización de la ayuda para estados de SQL desde el procesador de línea de mandatos

DB2 devuelve un valor de SQLSTATE para las condiciones que pueden ser el resultado de una sentencia de SQL. La ayuda de SQLSTATE explica los significados de los estados de SQL y los códigos de las clases de estados de SQL.

Para invocar la ayuda para estados de SQL, abra el procesador de línea de mandatos y entre:

```
? sqlstate o ? código de clase
```

donde *sqlstate* representa un estado de SQL válido de cinco dígitos y *código de clase* representa los dos primeros dígitos del estado de SQL.

Por ejemplo, `? 08003` visualiza la ayuda para el estado de SQL 08003, y `? 08` visualiza la ayuda para el código de clase 08.

Acceso a diferentes versiones del Centro de información de DB2

Para los temas de la versión 9.5 de DB2, el URL del Centro de información de DB2 es http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/

Para los temas de la versión 9 de DB2, el URL del Centro de información de DB2 es http://publib.boulder.ibm.com/infocenter/db2luw/v9/

Para los temas de la versión 8 de DB2, vaya al URL del Centro de información de la versión 8 en: http://publib.boulder.ibm.com/infocenter/db2luw/v8/

Visualización de temas en su idioma preferido en el Centro de información de DB2

El Centro de información de DB2 intenta visualizar los temas en el idioma especificado en las preferencias del navegador. Si un tema no se ha traducido al idioma preferido, el Centro de información de DB2 visualiza dicho tema en inglés.

- Para visualizar temas en su idioma preferido en el navegador Internet Explorer:
  1. En Internet Explorer, pulse en el botón **Herramientas** —> **Opciones de Internet** —> **Idiomas...**. Se abrirá la ventana Preferencias de idioma.
  2. Asegúrese de que su idioma preferido esté especificado como la primera entrada de la lista de idiomas.
     - Para añadir un nuevo idioma a la lista, pulse el botón **Agregar...**.

       Nota: La adición de un idioma no garantiza que el sistema tenga los fonts necesarios para visualizar los temas en el idioma preferido.

- Para mover un idioma hacia el principio de la lista, seleccione el idioma y pulse el botón **Subir** hasta que el idioma esté en primer lugar en la lista de idiomas.

3. Borre la antememoria del navegador y, a continuación, renueve la página para visualizar el Centro de información de DB2 en su idioma preferido.

- Para visualizar temas en su idioma preferido en un navegador Firefox o Mozilla:
  1. Seleccione el botón en la sección **Idiomas** del diálogo **Herramientas** —> **Opciones** —> **Avanzado**. Se visualizará el panel Idiomas en la ventana Preferencias.
  2. Asegúrese de que su idioma preferido esté especificado como la primera entrada de la lista de idiomas.
     - Para añadir un nuevo idioma a la lista, pulse el botón **Añadir...** a fin de seleccionar un idioma en la ventana Añadir idiomas.
     - Para mover un idioma hacia el principio de la lista, seleccione el idioma y pulse el botón **Subir** hasta que el idioma esté en primer lugar en la lista de idiomas.
  3. Borre la antememoria del navegador y, a continuación, renueve la página para visualizar el Centro de información de DB2 en su idioma preferido.

En algunas combinaciones de navegador y sistema operativo, puede que también tenga que cambiar los valores regionales del sistema operativo al entorno local y al idioma de su elección.

Actualización del Centro de información de DB2 instalado en el sistema o en el servidor de Intranet

Si ha instalado el Centro de información de DB2, puede descargar e instalar las actualizaciones que IBM pueda tener disponibles.

Para actualizar el Centro de información de DB2 instalado localmente es preciso que:

1. Detenga el Centro de información de DB2 en el sistema, y reinicie el Centro de información en modalidad autónoma. La ejecución del Centro de información en modalidad autónoma impide que otros usuarios de la red accedan al Centro de información, y permite descargar y aplicar actualizaciones.
2. Utilice la función Actualizar para ver qué actualizaciones están disponibles. Si hay actualizaciones que quisiera instalar, puede utilizar la función Actualizar para descargarlas y actualizarlas.

   Nota: Si su entorno requiere la instalación de actualizaciones del Centro de información de DB2 en una máquina no conectada a Internet, debe duplicar el sitio de actualizaciones en un sistema de archivos local utilizando una máquina que esté conectada a Internet y tenga instalado el Centro de información de DB2. Si muchos usuarios en la red van a instalar las actualizaciones de la documentación, puede reducir el tiempo necesario para realizar las actualizaciones duplicando también el sitio de actualizaciones localmente y creando un proxy para el sitio de actualizaciones.
   Si hay paquetes de actualización disponibles, utilice la característica Actualizar para descargar los paquetes. Sin embargo, la característica Actualizar sólo está disponible en modalidad autónoma.
3. Detenga el Centro de información autónomo y reinicie el Centro de información de DB2 en su equipo.

Nota: En Windows Vista, los mandatos listados más abajo se deben ejecutar como administrador. Para iniciar un indicador de mandatos o una herramienta gráfica con privilegios de administrador completos, pulse con el botón derecho del ratón el atajo y, a continuación, seleccione **Ejecutar como administrador**.

Para actualizar el Centro de información de DB2 instalado en el sistema o en el servidor de Intranet:

1. Detenga el Centro de información de DB2.
   - En Windows, pulse en **Inicio → Panel de control → Herramientas administrativas → Servicios**. A continuación, pulse con el botón derecho del ratón en el servicio **Centro de información de DB2** y seleccione **Detener**.
   - En Linux, especifique el mandato siguiente:
     ```
     /etc/init.d/db2icdv95 stop
     ```
2. Inicie el Centro de información en modalidad autónoma.
   - En Windows:
     a. Abra una ventana de mandatos.
     b. Navegue hasta la vía de acceso en la que está instalado el Centro de información. De forma predeterminada, el Centro de información de DB2 se instala en el directorio <Archivos de programa>\IBM\Centro de información de DB2\Versión 9.5, siendo `<Archivos de programa>` la ubicación del directorio Archivos de programa.
     c. Navegue desde el directorio de instalación al directorio doc\bin.
     d. Ejecute el archivo help_start.bat:
        ```
        help_start.bat
        ```
   - En Linux:
     a. Navegue hasta la vía de acceso en la que está instalado el Centro de información. De forma predeterminada, el Centro de información de DB2 se instala en el directorio /opt/ibm/db2ic/V9.5.
     b. Navegue desde el directorio de instalación al directorio doc/bin.
     c. Ejecute el script help_start:
        ```
        help_start
        ```

   Se inicia el navegador Web por omisión de los sistemas para visualizar el Centro de información autónomo.
3. Pulse en el botón Actualizar (). En la derecha del panel del Centro de información, pulse en Buscar actualizaciones. Se visualiza una lista de actualizaciones para la documentación existente.
4. Para iniciar el proceso de descarga, compruebe las selecciones que desea descargar, después pulse en Instalar actualizaciones.
5. Cuando finalice el proceso de descarga e instalación, pulse en Finalizar.
6. Detenga el Centro de información autónomo.
   - En Windows, navegue hasta el directorio doc\bin y ejecute el archivo help_end.bat:
     ```
     help_end.bat
     ```

     Nota: El archivo help_end de proceso por lotes contiene los mandatos necesarios para concluir sin peligro los procesos que se iniciaron mediante el archivo help_start de proceso por lotes. No utilice `Control-C` ni ningún otro método para concluir help_start.bat.
   - En Linux, navegue hasta el directorio de instalación doc/bin y ejecute el script help_end:

```
help_end
```

Nota: El script help_end contiene los mandatos necesarios para concluir sin peligro los procesos que se iniciaron mediante el script help_start. No utilice ningún otro método para concluir el script help_start.

7. Reinicie el Centro de información de DB2:
   - En Windows, pulse en **Inicio → Panel de control → Herramientas administrativas → Servicios**. A continuación, pulse con el botón derecho del ratón en el servicio **Centro de información de DB2** y seleccione **Iniciar**.
   - En Linux, especifique el mandato siguiente:
     ```
     /etc/init.d/db2icdv95 start
     ```

El Centro de información de DB2 actualizado visualiza los temas nuevos y actualizados.

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/db2help/.

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" en *pureXML Guide*
Configure una base de datos DB2 para almacenar datos XML y realizar operaciones básicas con el almacén de datos XML nativos.

"Visual Explain" en *Guía de aprendizaje de Visual Explain*
Analizar, optimizar y ajustar sentencias de SQL para obtener un mejor rendimiento al utilizar Visual Explain.

Información de resolución de problemas de DB2

Existe una gran variedad de información para la resolución y determinación de problemas para ayudarle en la utilización de productos DB2.

Documentación de DB2
Puede encontrar información sobre la resolución de problemas en la publicación DB2 Troubleshooting Guide o en la sección Soporte y resolución de problemas del Centro de información de DB2. En ellas encontrará información sobre cómo aislar e identificar problemas utilizando herramientas y programas de utilidad de diagnóstico de DB2, soluciones a algunos de los problemas más habituales y otros consejos sobre cómo solucionar problemas que podría encontrar en los productos DB2.

Sitio web de soporte técnico de DB2

Consulte el sitio Web de soporte técnico de DB2 si tiene problemas y desea obtener ayuda para encontrar las causas y 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), fix packs y otros recursos. Puede buscar en esta base de conocimiento para encontrar posibles soluciones a los problemas.

Acceda al sitio Web de soporte técnico de DB2 en la dirección http://www.ibm.com/software/data/db2/udb/support.html

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.

No puede descargar, exportar ni volver a exportar esta información excepto en el caso de cumplimiento total con todas las leyes y regulaciones vigentes, incluyendo todas las leyes y regulaciones sobre exportación de los 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

Es posible que IBM no comercialice en otros países algunos productos, servicios o características descritos en este manual. Consulte al representante local de IBM para obtener información sobre los productos y servicios que actualmente pueden adquirirse en su zona. Cualquier referencia a un producto, programa o servicio de IBM no pretende afirmar ni implicar que sólo se pueda utilizar dicho producto, programa o servicio de IBM. En su lugar se puede utilizar cualquier producto, programa o servicio funcionalmente equivalente que no vulnere ninguno de los derechos de propiedad intelectual de IBM. Sin embargo, es responsabilidad del usuario evaluar y verificar el funcionamiento de cualquier producto, programa o servicio que no sea de IBM.

IBM puede tener patentes o solicitudes de patentes en tramitación que afecten al tema tratado en este documento. La posesión de este documento no confiere ninguna licencia sobre dichas patentes. Puede realizar consultas sobre licencias escribiendo a:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
EE.UU.

Para realizar consultas sobre licencias referentes a información de doble byte (DBCS), puede ponerse en contacto con el Departamento de Propiedad Intelectual de IBM de su país/región o escribir a:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokio 106, Japón

El párrafo siguiente no es aplicable al Reino Unido ni a ningún país/región en donde tales disposiciones sean incompatibles con la legislación local:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROPORCIONA ESTA PUBLICACIÓN “TAL CUAL”, SIN GARANTÍA DE NINGUNA CLASE, NI EXPLÍCITA NI IMPLÍCITA, INCLUIDAS, PERO SIN LIMITARSE A ELLAS, LAS GARANTÍAS IMPLÍCITAS DE NO VULNERACIÓN DE DERECHOS, COMERCIALIZACIÓN O IDONEIDAD PARA UN FIN DETERMINADO. Algunos estados no permiten la exclusión de garantías expresas o implícitas en determinadas transacciones, por lo que es posible que esta declaración no sea aplicable en su caso.

Esta publicación puede contener inexactitudes técnicas o errores tipográficos. Periódicamente se efectúan cambios en la información aquí contenida; dichos cambios se incorporarán a las nuevas ediciones de la publicación. IBM puede efectuar, en cualquier momento y sin previo aviso, mejoras y cambios en los productos y programas descritos en esta publicación.

Este documento puede proporcionar enlaces o referencias a sitios y recursos que no son de IBM. IBM no representa, no da garantías, ni se compromete con los recursos de terceros ni con los recursos que no son de IBM a los cuales se puede hacer referencia, acceder desde o enlazarse con desde este documento. Un enlace a un sitio que no es de IBM no implica que IBM apruebe el contenido o la utilización de dicho sitio Web o a su propietario. Además, IBM no forma parte ni es responsable de ninguna transacción que el usuario pueda realizar con terceros, aún cuando llegue a conocerlos (o utilice un enlace a ellas) desde un sitio de IBM. De acuerdo a esto, el usuario reconoce y acepta que IBM no es responsable de la disponibilidad de dichos recursos o sitios externos ni tampoco es responsable de ningún contenido, servicio, producto u otros materiales que estén o se encuentren disponibles desde dichos sitios o recursos. Cualquier software que proporcionen terceras partes, estarán sujetos a los términos y condiciones de licencia que acompañen al software.

IBM puede utilizar o distribuir cualquier información que se le facilite de la manera que considere adecuada, sin contraer por ello ninguna obligación con el remitente.

Los licenciatarios de este programa que deseen obtener información sobre él con el fin de habilitar: (i) el intercambio de información entre programas creados de forma independiente y otros programas (incluido éste) y (ii) el uso mutuo de la información intercambiada, deben ponerse en contacto con:

IBM Canada Limited
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
CANADÁ

Dicha información puede estar disponible, sujeta a los términos y condiciones apropiados, incluido en algunos casos el pago de una tarifa.

El programa bajo licencia descrito en este documento y todo el material bajo licencia asociado a él, los proporciona IBM según los términos del Acuerdo de Cliente de IBM, el Acuerdo Internacional de Programas Bajo Licencia de IBM o cualquier acuerdo equivalente entre el usuario e IBM.

Los datos de rendimiento contenidos en este documento se obtuvieron en un entorno controlado. Por lo tanto, los resultados obtenidos en otros entornos operativos pueden variar significativamente. Algunas mediciones pueden haberse realizado en sistemas experimentales y no es seguro que estas mediciones sean las mismas en los sistemas disponibles comercialmente. Además, algunas mediciones pueden haberse calculado mediante extrapolación. Los resultados reales pueden variar. Los usuarios del presente manual deben verificar los datos aplicables para su entorno específico.

La información referente a productos que no son de IBM se ha obtenido de los proveedores de esos productos, de sus anuncios publicados o de otras fuentes disponibles públicamente. IBM no ha probado esos productos y no puede confirmar la exactitud del rendimiento, la compatibilidad ni ninguna otra afirmación referente a productos que no son de IBM. Las preguntas sobre las prestaciones de productos que no son de IBM deben dirigirse a los proveedores de esos productos.

Todas las declaraciones de intenciones de IBM están sujetas a cambio o cancelación sin previo aviso, y sólo representan objetivos.

Este manual puede contener ejemplos de datos e informes que se utilizan en operaciones comerciales diarias. Para ilustrarlos de la forma más completa posible, los ejemplos incluyen nombres de personas, empresas, marcas y productos. Todos estos nombres son ficticios y cualquier similitud con nombres y direcciones utilizados por una empresa real es totalmente fortuita.

LICENCIA DE COPYRIGHT:

Este manual puede contener programas de aplicaciones de ejemplo escritos en lenguaje fuente, que muestran técnicas de programación en diversas plataformas operativas. Puede copiar, modificar y distribuir estos programas de ejemplo como desee, sin pago alguno a IBM con la intención de desarrollar, utilizar, comercializar o distribuir programas de aplicaciones de acuerdo con la interfaz de programación de aplicaciones correspondiente a la plataforma operativa para la que están escritos los programas de ejemplo. Estos ejemplos no se han probado exhaustivamente bajo todas las condiciones. Por lo tanto, IBM no puede asegurar ni implicar la fiabilidad, utilidad o función de estos programas.

Cada copia o parte de estos programas de ejemplo o cualquier trabajo derivado debe incluir una nota de copyright como la siguiente:

© (*nombre de la empresa*) (*año*). Partes de este código proceden de programas de ejemplo de IBM Corp. © Copyright IBM Corp. *_entre el o los años_*. Reservados todos los derechos.

Marcas registradas

Los nombres de empresas, productos o servicios identificados en los documentos de la biblioteca de documentación de DB2 Versión 9.5 pueden ser marcas registradas o marcas de servicio de International Business Machines Corporation o de otras empresas. La información sobre marcas registradas de IBM Corporation en los Estados Unidos y/o en otros países está ubicada en http://www.ibm.com/legal/copytrade.shtml.

Los términos siguientes son marcas registradas de otras empresas y se han utilizado como mínimo en uno de los documentos de la biblioteca de documentación de DB2:

Microsoft, Windows, Windows NT y el logotipo de Windows son marcas registradas de Microsoft Corporation en los Estados Unidos y/o en otros países.

Intel, el logotipo de Intel, el logotipo de Intel Inside, Intel Centrino, el logotipo de Intel Centrino, Celeron, Intel Xeon, Intel SpeedStep, Itanium y Pentium son marcas registradas de Intel Corporation en los Estados Unidos y/o en otros países.

Java y todas las marcas registradas basadas en Java son marcas registradas de Sun Microsystems, Inc. en los Estados Unidos y/o en otros países.

UNIX es una marca registrada de The Open Group en los Estados Unidos y/o en otros países.

Linux es una marca registrada de Linus Torvalds en los Estados Unidos y/o en otros países.

Adobe, el logotipo de Adobe, PostScript y el logotipo de PostScript son marcas registradas o marcas comerciales de Adobe Systems Incorporated en los Estados Unidos y/o en otros países.

Otros nombres de empresas, productos o servicios, pueden ser marcas registradas o marcas de servicio de otras empresas.

Índice

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

X

SC11-3501-00

Spine information:

DB2 Versión 9.5 for Linux, UNIX, and Windows **Versión 9 Release 5** **Desarrollo de aplicaciones Java**